IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Microservices backlog – a genetic programming technique for identification and evaluation of microservices from user stories.

Fredy H. Vera-Rivera[1,2,4], Eduard Puerto[1], Hernán Astudillo[3], and Carlos Gaona[4]

[1]Grupo de Investigación GIA, Universidad Francisco de Paula Santander, San José de Cúcuta, Norte de Santander, COLOMBIA
[2]Foundation of Researchers in Science and Technology of Materials - FORISTOM, Bucaramanga, COLOMBIA
[3]Grupo de Investigación TOESKA, Universidad Técnica Federico Santa María, Valparaíso, CHILE
[4]Grupo de Investigación GEDI, Universidad del Valle, Santiago de Cali, COLOMBIA

Corresponding author: First F.H. Vera-Rivera (e-mail: fredyhumbertovera@ufps.edu.co).

This work was partially supported by Colombia's Ministry of Science and Technology (Minciencias - Colciencias) through doctoral scholarship "753 - Formación de capital humano de alto nivel para el departamento Norte de Santander"; by the Francisco de Paula Santander University (Cúcuta, Colombia) through the doctoral studies commission number 14 of 2016; by the Universidad del Valle (Cali, Colombia); and by ANID (Chile) through PIA/APOYO AFB180002.

ABSTRACT The microservice granularity directly affects the quality attributes and usage of computational resources of the system, determining optimal microservice granularity is an open research topic. Microservices granularity is defined by the number of operations exposed by the microservice, the number of microservices that compose the whole application, and its complexity and dependencies. This paper describes "Microservice Backlog (MB)", a semiautomatic model for defining and evaluating the granularity of microservice-based applications; MB uses genetic programming technique to calculate at design time the granularity of each microservice from the user stories in the "product backlog" or release planning; the genetic algorithm combined coupling, cohesion, granularity, semantic similarity, and complexity metrics to define the number of microservices, and the user stories associated with each microservice. MB decomposes the candidate microservices, allowing to analyze graphically the size of each microservice, as well as its complexity, dependencies, coupling, cohesion metrics, and the number of calls or requests between microservices. The resulting decomposition (number of microservices and their granularity) performed by MB shows less coupling, higher cohesion, less complexity, fewer user stories associated with each microservice, and fewer calls among microservices. MB was validated against three existing methods, using two state-of-the-art applications (Cargo Tracking and JPet-Store), and one real-life applications (Foristom Conferences). The development team and/or architect can use metrics to identify the critical points of the system and determine at design time how the microservice-based application will be implemented.

INDEX TERMS Service-oriented systems engineering, Service computing, Software design, Software architecture, Web services, Micro-services granularity, Microservices decompositions, Genetic algorithms, Software metrics.

I. INTRODUCTION

The complexity involved in software development has been addressed with the use of agile methodologies and practices that were born from the agile manifesto and its principles, in contrast to traditional forms [1]. Last years, software companies have been practicing agile development methods [2]; according to the 14th annual state of agile report [3], accelerating software delivery and enhancing ability to manage changing priorities remain the top reasons stated for adopting agile; the more used agile techniques were daily standup, retrospectives, sprint/iteration planning, sprint/iteration review, and short iterations; the most engineering practices employed were unit testing, coding standards, continuous integration, refactoring, and continuous delivery. Sprint/iteration planning is usually done in a product backlog, which lists the functional requirements of the application as user stories, along with their priorities and estimated time and story points [4]. The microservices architecture facilitates permanent, faster, and automated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

updates using DevOps practices, achieving shorter, automated, widely tested deliveries, and refactoring [5].

Microservices are single-responsibility units (granules) that encapsulate data and processing logic, they are deployed remotely; these remote units are services that can be deployed, changed, substituted, and scaled independently of each other [6]. The quality of a microservice-based system is influenced by the granularity of its microservices since their size and number directly affect the system's quality attributes. The optimal size or granularity of a microservice directly affects application performance, maintainability, storage (transactions and distributed queries), and usage and consumption of computational resources (mainly in the cloud, the usual platform to deploy and execute microservices) [7]. Although the size of microservice or optimal granularity is a discussion topic, few patterns, methods, or models exist to determine how small a microservice should be.

Hassan *et al.* [8] stated that a granularity level determines "the service size and the scope of functionality a service exposes [9]". Granularity adaptation entails merging or decomposing microservices thereby moving to a finer or more coarse-grained granularity level. Homay *et al.* [10] stated that "the problem in finding service granularity is to identify a correct boundary (size) for each service in the system. In other words, each service in the system needs to have a concrete purpose, as decoupled as possible, and add value to the system. A service has a good granularity if it maximizes system modularity while minimizing the complexity. Modularity in the sense of flexibility, scalability, maintainability, and traceability, whereas complexity in terms of dependency, communication, and data processing".

The definition of microservices granularity is presented in the following problem context, first in migrations from monolith to microservices or decompositions, second in the development of microservices-based applications from scratch, and third in the development of microservices-based applications composing existing services. The migrations from monolith to microservices have been widely studied, migrations have a great interest to both academia and industry, while the other two approaches have been studied very few [11].

The problem addressed in this research focused on the design of microservices-based applications from scratch, which begins when the development team or architect, after performing an analysis, determines that the application needs to be implemented using the microservices architecture, in the context of agile software development. The development team establishes the functional requirements as user stories in the product backlog, establishing its priorities and estimates; from the product backlog, the development team needs to identify the number of microservices to be implemented and associate the user stories to each microservice maintaining low coupling, high cohesion, and low complexity among microservices.

Furthermore, we introduce the Microservices Backlog (MB), a model that allows software architects or development team to graphically analyze the microservices granularity; MB focus on three relevant activities: 1) Determining and evaluating the granularity of microservices, 2) establishing the number of user stories assigned to each microservice, and 3) establishing the optimal number of microservices that will be part of the application. These activities will support microservices' low coupling, high cohesion, and low complexity properties. Design time metrics were adapted and calculated to evaluate decomposition or microservice-based applications.

MB was evaluated in three projects, two state-of-the-art case studies (Cargo Tracking and JPet-Store) and one real-life case studies (Foristom Conferences). Comparing the proposed decomposition against domain-driven design (DDD) and state-of-the-art methods; MB yields microservices-based applications with lower coupling, less complexity, less communication, and dependencies among microservices, fewer user stories associated with a microservice, and higher semantic coherence among the user stories in a microservice.

We have been working on this problem, in [12], a first approximation of the MB was proposed, which used a genetic algorithm with coupling, cohesion, and granularity metrics; this genetic algorithm did not consider the semantic similarity between user stories and microservices, it did not use complexity metrics. This paper extends that work considerably including analysis of the semantic similarity among entities of the user stories and microservices, a cognitive complexity metric to evaluate decompositions was proposed, and additional validations (initially with Cargo Tracking application, now also with Jpet-Store and the real-life case studies Foristom Conferences.

The main contributions from this work were: 1) a model for determining and evaluating the granularity of microservices at design time, establishing the number of user stories assigned to a microservice and the number of microservices that are part of the application, ensuring that microservices have low coupling, low complexity, high cohesion, and fewer dependencies; 2) identified and adapted metrics of complexity, coupling, cohesion, size of the microservice, development time, and calls between microservices; 3) mathematical formalization of a microservice-based application in terms of user stories and metrics, and 4) we update the previous genetic algorithm to assign user stories to microservices, including semantic similarity and complexity; although MB can also be used in migrations.

The remainder of this paper is organized as follows, section II related works; section III Methodology and evaluation methods used; section IV our approach; section V discussing results; section VI limitations and future works and Section VII, summarizes our conclusions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

II. RELATED WORKS

We identified several methods, methodologies, and techniques to determine microservices granularity through a systematic literature review [11]. The most used techniques included machine learning clustering, semantic similarity, genetic programming, and domain engineering. Table I details the papers by year compared with our approach.

Additionally, Service Cutter is a method and tool framework for service decomposition [13], in it, coupling information is extracted from software engineering artifacts. This approach is more appropriate for SOA applications, but it has been used for comparative analysis in the surveyed works.

Other authors proposed patterns to address microservice development, such as Richardson [14] proposed decomposition patterns, Zimmermann *et al.* [15] proposed a microservice API patter (MAP) for API design and evolution, with five categories: (1) foundation, (2) responsibility, (3) structure, (4) quality, and (5) evolution. These patterns are an important reference for developing microservice-based applications. However, there is no specific pattern to determine the number and size of microservices.

TABLE I

RELATED WORKS TO THE MICROSERVICE GRANULARITY PROBLEM

Year	Papers	Metrics	Quality attributes	Technique, method, or methodology description	Input data
2020	Microservice Backlog – Our approach	Complexity, coupling, cohesion, granularity, performance: microservices calls.	Modularity. Maintainability. Functionality. Performance.	Genetic algorithm. Semantic similarity (Natural processing language).	User stories
2020	2 [16], [17].	Cohesion, granularity.	None	- Domain-driven design, architectural design via dynamic software visualization. - Clustering using affinity propagation algorithm, and clustering of semantically similar	- Source code - Runtime logs
2019	12 [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29].	Coupling, cohesion, granularity, computational resource, performance, source code,	Scalability. Performance. Functionality. Modularity. Maintainability.	- Machine learning, scale weighted k-means. - Dataflow-driven decomposition algorithm. - Process-mining approach, DISCO used to identify the business processes. - Search-based functional atom grouping algorithm. Non-dominated sorting genetic algorithm-II. - Set of rule-based decisions, adaptation of the four-step rule set (4SRS) method. - Word embedding and hierarchical clustering of semantic similarity. - Microservice discovery algorithms. - clustering algorithm applied to aggregate domain entities. - Service granularity cost analysis-based method, cost analysis function. - Validation framework for microservice decompositions. - Ontology scheme search-based techniques, multi-objective genetic algorithm. - Non-dominated sorting genetic algorithm-II (NSGA II).	- Access logs - Dataflow diagram, - Use cases. - Execution logs. - Execution traces from logs. - OpenApi specification - Source code - Database - Execution call graphs- - Component and microservices properties.
2018	6 [30], [31], [32], [33], [34], [35].	Coupling, cohesion, complexity, granularity, computational resource, performance,	Scalability. Performance. Availability.	- Domain engineering, domain-driven design. - Domain-driven design COSMIC function points. - Functional decomposition. - Heuristics used for functional splitting, microservice discovery algorithms. - Decomposition pattern.	- Use cases - Source code, database, execution call graphs. - Scenario statements, workflow, BPEL description.
2017	7 [36], [37], [38], [39], [40], [41], [42]	Performance.	Scalability. Performance. Reliability. Maintainability	- Vertical decomposition in self-contained systems. - Balance cost quality assurance vs deployment. - Comparing the same microservices in a single container and in two containers. - Architecture definition language (ADL). - Semantic similarity, clustering k-means, DISCO. - Graph-based clustering algorithm. - Virtual machine image synthesis and analysis.	- OpenApi specification. - Source code.
2016	2 [43], [44].	Coupling, security, and scalability impact.	Scalability. Security	Self-adaptative solution. Decomposition from system requirements – security vs scalability	None

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

None of the reviewed works used agile software development artifacts as inputs, (i.e. user stories, product backlog, release planning, Kanban boards) to define or assess microservices' granularity.

The most addressed quality attributes in the reviewed papers were scalability and performance (runtime characteristics), and modularity and maintainability (software artifact characteristics) were the least addressed. Only one paper [18] addressed both runtime and software artifact characteristics. No papers addressed functionality, performance, modularity, and maintainability at the same time.

Some papers use metrics to evaluate microservices granularity, including coupling, cohesion, number of calls, number of requests, and response time, although few methods or techniques use complexity as a metric: thus, [25] used Number of singleton clusters and maximum cluster size, and [31] used COSMIC function points (Common Software Measurement International Consortium). Cognitive complexity was not considered by related works.

III. METHODOLOGY AND EVALUATION METHODS

We used design science research, following the paradigm of Hevner *et al.* [45] The design-science paradigm seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts (see figure 1). The proposed artifact was the Microservice backlog model.

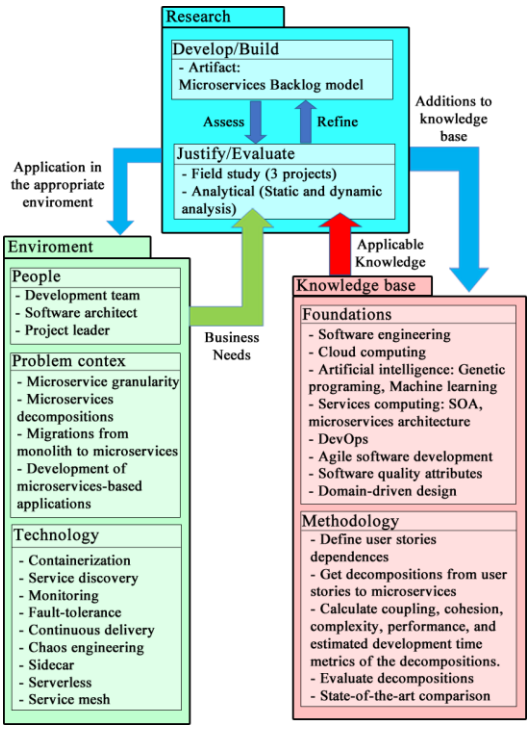


FIGURE 1. Research model. Design science research framework. Adapted from Hevner et al. [45].

The research process began with the design and development of MB, which was iteratively evaluated in a field study through a static and dynamic analysis; with each evaluation, it was improved and corrected until obtaining an optimal proposal. The construction of MB is based on the following theoretical foundations: software engineering, artificial intelligence, cloud computing, service computing, and agile software development.

The stakeholders were software architects, software development teams, and project leaders, who want to develop or migrate a microservice-based application; the microservice backlog model allows them to define the microservice granularity and evaluate the application architecture.

The research process is detailed below:

1. **Problem context definition:** we defined the problem context: microservices granularity, microservices decompositions and migrations from monolith to microservices, and development of microservices-based applications.
2. **Theoretical foundations and state of the art:** we performed a systematic literature review, identified, adapted, and proposed metrics for defining the microservices granularity; and identified the related works.
3. **Design MB:** We design the Microservices Backlog and proposed a formal specification of the granularity model.
4. **Develop MB:** We built the intelligent granularity model and implemented the genetic algorithm technique to decompose the product backlog into microservices. We implemented an algorithm to evaluate metrics for microservices decompositions or microservice-based applications.
5. **Evaluation of MB:** We evaluated the model using state-of-the-art examples (Cargo Tracking and JPet Store) and one real-life project (Foristom Conferences). The evaluation compared decomposition yield by MB versus decompositions by other methods: Domain-driven design (DDD) [46], Service Cutter [13], Microservices Identification Through Interface Analysis (MITIA) [47], and Service Candidate Identification from Monolithic Systems based on Execution Traces (Execution Traces) [18]. We took the decompositions proposed by MITIA and Execution traces about state-of-the-art examples, next we identified the operations associated with each microservice, then the operations were associated with user stories. Traditionally, the user stories specify the functional requirements of the application, the user stories are implemented as operations.
   Since DDD is the most widely used method for microservices identification, the evaluation of the real-life project verified that the obtained decomposition was consistent and close to DDD.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

6. **Propose MB:** Based on metrics and analytical evaluation including adjustment through the research process, then MB was proposed as an intelligent specification and granularity evaluation model.

A. EVALUATION METHODS

As recommended by Hevner *et al.* [45] we used observational and analytical evaluation methods to assess MB. The observational method was a field study, which we used and monitored the microservice backlog model in three projects, the projects are detailed in section V.

The analytical methods were both static and dynamic analysis. We calculated metrics of complexity, coupling, cohesion, dependencies, performance, and size of the proposed decomposition (or microservice-based application), then we compared it with other approaches. The metrics were calculated from the user stories data and their dependencies at design time.

We carried out the evaluation process as follow:

1. We analyzed and described the state-of-the-art examples and the real-life project.
2. The user stories of each project were specified.
3. We defined the dependencies among user stories. Which were identified according to the business logic, dataflow, invocations, or calls between operations or uses stories.
4. We got the decomposition through MB and the decompositions of the state-of-the-art approaches.
5. For each decomposition, the metric calculator algorithm calculated the metrics and draw the graph or diagram.
6. We evaluated the decompositions and compared the metrics.

The evaluation aim was to verify that MB allowed to define the appropriate granularity of the microservices and to compare the cognitive complexity, coupling, cohesion, and dependencies of the decompositions.

IV. MICROSERVICES BACKLOG

Microservice Backlog is a model (see figure 2), designed to graphically analyze the microservices granularity, starting from a set of functional requirements expressed as user stories within a product backlog (prioritized and characterized list of functionalities that an application must contain). The model specifies the architecture of microservices-based applications. After this, the architect or development team can evaluate the appropriate granularity or size of each microservice considering some characteristics such as complexity, coupling, cohesion, development time, and use of computational resources at design time. This way, the architect or developer can find a strategy for its implementation.

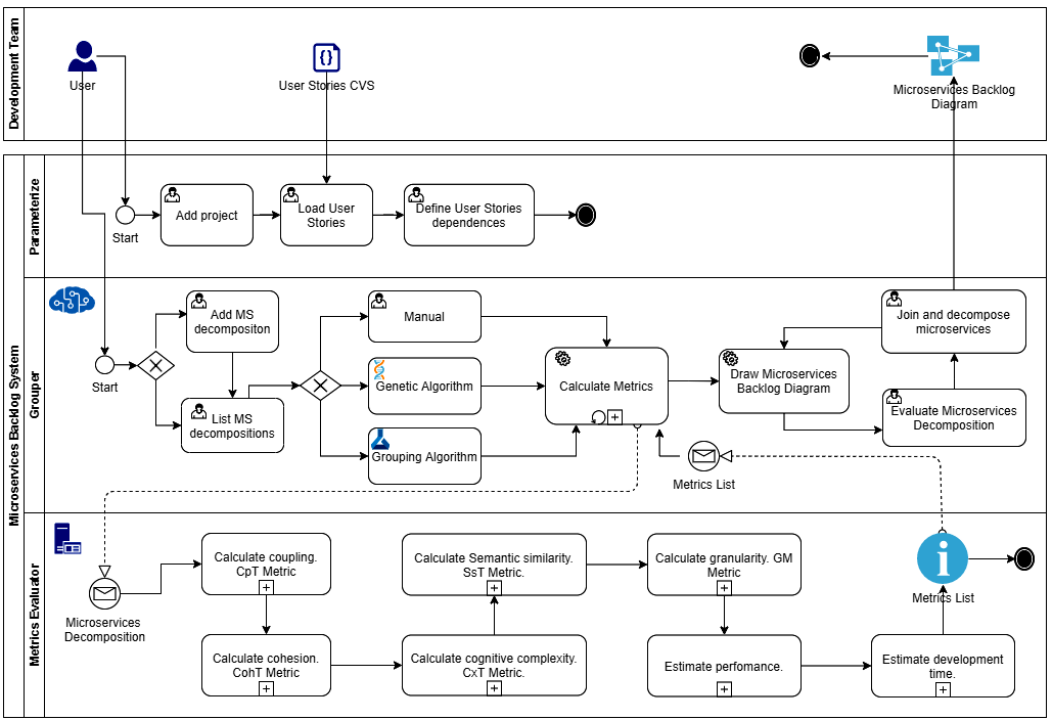


FIGURE 2. **Microservices Backlog model. Semi-automatic decomposition from user stories to microservices.**

MB was implemented in a web application (Django - Python) and consists of the following components (see figure 2):

A. Formal specification of the granularity model.
B. Parameterize component.
C. Grouping component, which implements the grouping techniques.
D. Metric calculator component.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

E. Microservices Backlog diagram and decomposition evaluator.

A. FORMAL SPECIFICATION OF THE GRANULARITY MODEL

The formal specification corresponds to the mathematical expressions that allow to calculate the metrics and to evaluate the objective function of the granularity model. The formal specification is given in terms of the metrics of coupling *(CpT)*, cohesion *(CohT)*, number of stories associated to the microservice *(WsicT)*, cognitive complexity *(CxT)*, semantic similarity *(SsT)*, and the granularity metric *(Gm)*. Additionally, for the evaluation process, metrics of complexity (*P:* story points), communication, performance and estimated development time are included.

According to Hassan *et al.* [8] and Homay *et al.* [10] microservice granularity definition (see introduction), a relationship between the microservice granularity with coupling, modularity and complexity of the system is evident. Changing the size and scope of microservices implies changes in the coupling, modularity, and complexity of the system. In this case, granularity corresponds to defining the number of user stories associated with a microservice (service size) and the number of microservices that comprising the application (application size).

A microservice can have a low granularity (smaller size), but when interacting with other microservices the coupling increases, if the coupling is very high, it is a bad decision to maintain that granularity in the microservice-based system, then the microservice should be joined with other microservices to reduce the coupling, when joining with other microservices its granularity increases, because it will have more associated operations to expose. With the proposed model, we seek to determine the appropriate granularity in such a way that its coupling is low, that it has few dependencies and little communication with other microservices, being consistent with the theoretical definition.

Microservices should have a specific purpose, therefore, services / operations / stories that refer to the same purpose should be grouped in the same microservice, hence the importance of semantic similarity, if user stories that refer to the same thing, which have a high semantic similarity should be grouped in the same microservice, thus being an indicator of high cohesion.

The specification formal of the granularity model will be given in terms of the metrics and the granularity metric *(Gm)*. Let microservice-based application *(MSBA)* as:

$$MSBA = (MS, \overrightarrow{MT}) \quad (1)$$

Where *MS* is a set of microservices, $MS = \{ms_1, ms_2, \ldots, ms_n\}$ and $\overrightarrow{MT}$ is a vector of the metrics calculated for *MSBA*.

$$\overrightarrow{MT} = [CpT, CohT, WsicT, CxT, (100 - SsT)] \quad (2)$$

Where *CpT* is the coupling, *CohT* is the cohesion, *WsicT* is the greater number of user stories associated with a microservice, *CxT* is the cognitive complexity points, and *SsT* is the semantic similarity, which are metrics for *MSBA*. These metrics were adapted from state-of-the-art approaches [48], [49], and [50]. We proposed the cognitive complexity points as a complexity metric.

SsT corresponds to the value of the semantic similarity obtained by the Spacy library, which is a value between zero and one, the closer to one is, the greater semantic similarity it has; for this model, we amplify the similarity value, it is a number between 0 and 100 (percentage) in such a way that its dimension is like the dimension of the other variables. Equation (2) included *(100 - SsT)* to invert its relationship, having greater semantic similarity when similarity is close to 0; then, this expression is used to calculate *Gm* and is minimized in the objective function of the genetic algorithm.

The microservices has associated user stories and metrics, then:

$$ms_i = (HU_i, MTS_i) \quad (3)$$

Where ms_i is the *i-th* microservice, HU_i is the set of user stories associated with the *i-th* microservice, then $HU_i = \{hu_1, hu_2, \ldots, hu_m\}$. MTS_i is a set of metrics calculated for ms_i.

1. COUPLING OF MSBA (*CpT*)

The coupling determines the degree of dependence of one software component with another. Coupling is defined by three metrics: absolute importance of the microservice *(AIS)*, absolute dependence of the microservice *(ADS)*, and microservices interdependence *(SIY)*. These metrics are calculated based on the dependencies of the user stories for each microservice.

The absolute importance of the microservice (AIS): AIS is the number of other microservices that invoke at least one operation of a microservice's interface [50]. AIS_i is the number of clients invoking at least one operation of MS_i. At the system level, the $\overrightarrow{AIS}$ vector is defined, which contains the calculated *AIS* value for each microservice.

To calculate the total value of *AIS* at the system level *(AisT)*, the vector norm is calculated. Where *n* is the number of microservices of the MSBA, thus:

$$\overrightarrow{AIS} = [AIS_1, AIS_2, \ldots, AIS_n] \quad (4)$$

$$AisT = \left|\overrightarrow{AIS}\right| = \sqrt[2]{AIS_1^{\,2} + AIS_2^{\,2} + \ldots + AIS_n^{\,2}} \quad (5)$$

The absolute dependence of the microservice (ADS): ADS is the number of other microservices that microservice depends on. The number of microservices from which invokes at least one operation [50]. ADS_i is the number of other microservices on which the MS_i depends. To calculate the total value of *ADS* at the system level *(AdsT)* the $\overrightarrow{ADS}$ vector norm is calculated. Then:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

$$\overrightarrow{ADS} = [ADS_1, ADS_2, \ldots, ADS_n] \quad (6)$$

$$AdsT = \left|\overrightarrow{ADS}\right| = \sqrt[2]{ADS_1{}^2 + ADS_2{}^2 + \ldots + ADS_n{}^2} \quad (7)$$

Microservice interdependence (SIY): SIY is the number of interdependent microservices pairs [50]. *SIY* defines the number of pairs of microservices that depend bi-directionally on each other divided by the total number of microservices. At the system level, the vector $\overrightarrow{SIY}$ was defined:

$$\overrightarrow{SIY} = [SIY_1, SIY_2, \ldots, SIY_n] \quad (8)$$

$$SiyT = \left|\overrightarrow{SIY}\right| = \sqrt[2]{SIY_1{}^2 + SIY_2{}^2 + \ldots + SIY_n{}^2} \quad (9)$$

Let the $\overrightarrow{Cp}$ vector as the *MSBA* level coupling metric, calculating the norm of the vector $\overrightarrow{Cp}$ we have the coupling value for the application *(CpT)*:

$$\overrightarrow{Cp} = [AisT, AdsT, SiyT] \quad (10)$$

$$CpT = 10 * \left|\overrightarrow{Cp}\right| = 10 * \sqrt[2]{AisT^2 + AdsT^2 + SiyT^2} \quad (11)$$

We amplify *CpT* by 10, in such a way that its dimension is like the dimension of the other variables of $\overrightarrow{MT}$.

Figure 3 shows an example of the coupling metric calculation for a hypothetical case in which there are 3 microservices forming *MSBA*. As follows $ms_1 = \{hu_1, hu_2\}$, $ms_2 = \{hu_3\}$ and $ms_3 = \{hu_4\}$. Where hu_1 has as dependencies $\{hu_3, hu_4\}$, hu_2 has $\{hu_4\}$, hu_3 has $\{hu_1\}$ and hu_4 has no dependencies.

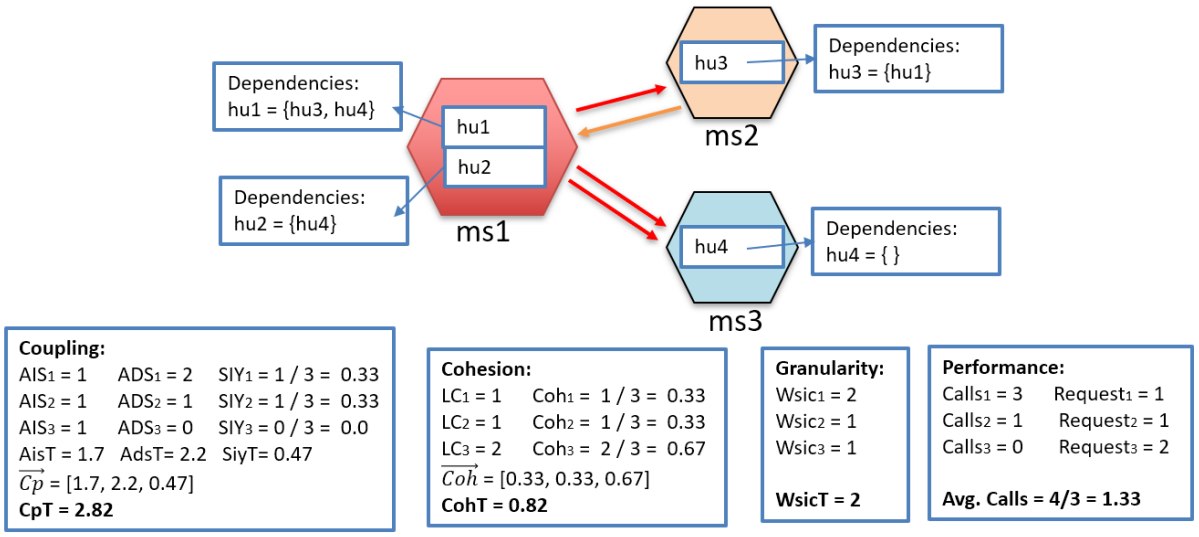


FIGURE 3. Example of metrics calculation.

2. COHESION OF MSBA (CohT)

Cohesion and coupling are two contrasting properties. A solution balancing high cohesion and low coupling. We used the lack of cohesion (*LC*), lack of cohesion grade *(Coh)*, and semantic similarity *(SsT)* for measuring the cohesion of *MSBA*.

Lack of cohesion (LC): *LC* measured the number of pairs of microservices not having any dependency between them, adapted from [49]. *LC* of MS_i was defined by us as the number of pairs of microservices not having any interdependency between MS_i.

Lack of cohesion grade (Coh): The degree of cohesion *Coh* of each microservice is defined as the proportion of the lack of cohesion metric divided by the total number of microservices that are part of the application.

$$Coh_i = LC_i / n \quad (12)$$

Where *n* is the number of microservices. At the system level, the vector $\overrightarrow{Coh}$ was defined, calculating the norm of the vector, we have the cohesion grade for the application *(CohT)*:

$$\overrightarrow{Coh} = [Coh_1, Coh_2, \ldots, Coh_n] \quad (13)$$

$$CohT = \left|\overrightarrow{Coh}\right| = \sqrt[2]{Coh_1{}^2 + Coh_2{}^2 + \ldots + Coh_n{}^2} \quad (14)$$

Figure 3 shows the cohesion metric calculation example for the hypothetic case.

Semantic Similarity of MSBA (SsT): According to Cojocaru et al. [27] "semantic similarity uses lexical distance assessment algorithms to flag the services that contain unrelated components or unrelated actions hindering cohesion".

SsT was calculated using the natural process language library Spacy [51], in which the similarity is determined by comparing word vectors or "word embeddings", multi-dimensional meaning representations of a word. We calculated the semantic similarity between each user story,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

joining the name and the description of the user story. We calculated *SsT* as follow:

1. We selected the nouns from the name and description of the user story.
2. We identified the lemmas of the noun in each user story.
3. We defined a dictionary that contains the semantic similarity values among the user stories as follow:

$$DSS = \{< "hu_1 - hu_2", a_{1-2} >, < "hu_1 - hu_3", a_{1-3} >, \ldots, < "hu_j - hu_k", a_{j-k} >\} \quad (15)$$

Where:

"hu_j - hu_k" is the dictionary´s key, which is formed by the concatenation of the user stories' identifiers.

a_{j-k} is the dictionary´s value, which corresponds to the semantic similarity value obtained by spacy among user stories *j* and *k*, it is a float number between 0 and 1.

4. We calculated the semantic similarity *(SS_i)* of MS_i as the average of the semantic similarity values between its user stories. The total semantic similarity of *MSBA* was the semantic similarity average of the microservices. For obtaining a value of semantic similarity between 0 and 100 we multiplied the average by 100.

$$SS_i = \frac{1}{c} \sum_{j=1,k=j+1}^{m} a_{j-k} \quad (16)$$

$$SsT = \frac{100}{n} \sum_{i=1}^{n} SS_i \quad (17)$$

Where:

m is the number of user stories of the *i-th* microservice.

c is the number of comparisons done to calculate *SS;* it is the number of combinations between the microservice user stories.

n is the number of microservices of *MSBA.*

3. GRANULARITY OF MSBA (WsicT)

The granularity corresponds to the size of each microservice and the size of the application. We used the granularity metrics listed below.

The number of microservices (n): The number of microservices that are part of the system or *MSBA*.

Weighted service interface count (WSIC): *WSIC* is the number of exposed interface operations of MS_i [52]. For our model, a user story is related to an operation (one-to-one); so, we adapt this metric as the number of user stories associated with the microservice. Other authors called this metric the operation number. We adapt *WSIC* as the number of user stories assigned to each microservice. We defined *WsicT* as the maximum number of user stories associated with a microservices, so *WsicT* is the maximum *WSIC* of *MSBA*, then

$$WsicT = Max(WSIC_1, WSIC_2, \ldots, WSIC_n) \quad (18)$$

Also, figure 3 illustrates the calculation of WsicT.

4. PERFORMANCE

Estimating the performance of an application at design time is difficult and imprecise. We used the number of calls and requests between microservices for estimating the performance.

We assume that if there are more calls and requests between the microservices, then the communication, latency, and response time of the application is increased, therefore the performance of the application is directly affected. The aim may be to have microservices that do not have communication between them and work independently. Therefore, we define two metrics:

Calls of a microservice *($Calls_i$)*: Calls corresponds to the number of invocations of MS_i to another microservices of *MSBA*.

Requests of a microservice *($request_i$)*: Request corresponds to the number of invocations of other microservices to MS_i of *MSBA.*

Average of calls of MSBA (Avg. Calls): *Avg. Calls* are the average of calls among microservices of MSBA.

$$Avg.Calls = \frac{1}{n} \sum_{i=1}^{n} Calls_i \quad (19)$$

Where:

n is the number of microservices of *MSBA.*

Figure 3 presents an example of the calculation of the request and calls for a MSBA.

5. COMPLEXITY OF MSBA (CxT)

Measuring complexity is fundamental for developing microservice-based applications. If the complexity is high, then the cost of change is higher. So, the used complexity metrics are detailed below.

User story points (P): The user story points are an estimated point of the effort needed to develop the user story. The story points are an indicator of the speed of development of the team; then we defined P_i as the user story points of MS_i as follow:

$$P_i = \sum_{j=1}^{m} PH_j \quad (20)$$

Where:

P_i is the total of user story points of MS_i.

m is the number of user stories of MS_i.

PH_j is the estimated user story points of j-th user story of MS_i.

Cognitive complexity points (*CxT*): We proposed a metric of cognitive complexity points (*CxT)* as follows: Points were added according to the complexity of the microservice, its relationships, and dependencies. The difficulty of developing and maintaining a microservice-based application was estimated. The starting point was the estimation of story points made by the development team.

CxT was based on the complexity of a graph and its depth. We started from a base case, which corresponds to the least complexity. This case would be a single microservice, with one user story and one estimated story point for its

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

development. For this case $Cx_0 = 2$. *CxT* corresponds to the number of times that the application is more complex in relation to the base case. Formally *CxT* was defined as follows:

$$Cx = \left(\left(\sum_{i=1}^{n} Cg_i\right) + Max(P_1, \ldots, P_n) + (n * WsicT) + \left(\sum_{i=1}^{n} Pf_i\right) + \left(\sum_{i=1}^{n} SIY_i\right)\right) \quad (21)$$

$$CxT = \frac{Cx}{Cx_0} \quad (22)$$

Where:
CxT = Cognitive complexity points of *MSBA*.
i = *i-th* microservice
$Cg_i = P_i * (Calls_i + Request_i)$, $Calls_i$ are the outputs of MS_i and $Request_i$ are the inputs of MS_i.
P_i = Total user story points of MS_i. According to (20)
$Max(P_1, \ldots, P_n)$: Maximum P_i of *MSBA*.
n = number of microservices of *MSBA*.
WsicT: Greater WSIC of the application. According to (18)
Pf_i: Number of nodes used sequentially from a call that makes a microservice to other microservices, counted from the i-th microservice; A larger depth implies a greater complexity of implementing and maintaining the application.
SIY: Microservice Interdependence.
Cx_0: The base case where the application has one microservice, one user story with one estimated story point. Then $Cg_1 = 0$, $Greater(P_1) = 1$, $n=1$, $WsicT=1$, $Pf_1 = 0$, $SIY=0$, and $Cx = 2$. Therefore $Cx_0 = 2$.

6. ESTIMATED DEVELOPMENT TIME OF MSBA (T)

The microservices are implemented and organized around business capabilities; ideally, each one is managed by an independent development team. For the evaluations of this model, we assumed that each microservice is developed in parallel and independently; thus, the estimated development time of the application corresponds to the longest estimated development time of the microservices that are part of *MSBA*. In real life this is not entirely true, a development team oversees several microservices and several microservices are developed sequentially; this restriction will be considered in future work.

The development team estimates the user story points and the development time in the release planning. Many software development companies define a scale of conversion of user story points to development time (hours). We assumed that de estimated development time of the user stories as an input data of this model. We defined two evaluation metrics as follows.

Microservice's development time (t_i): The microservice's development time corresponds to the sum of the estimated development time of each user story that is part of the microservice.

$$t_i = \sum_{j=1}^{m} TH_j \quad (23)$$

Where:
t_i is the estimated development time of MS_i.
m is the number of user stories of MS_i.
TH_j is the estimated development time of the *j-th* user story of MS_i., it is an input data of the model.

Application development time (T): Greater estimated development time of the microservices that are part of *MSBA*.

$$T = Max(t_1, t_2, \ldots, t_n) \quad (24)$$

7. GRANULARITY METRIC OF MSBA (Gm)

Finally, the value of the target function *Gm* use (2), *Gm* is defined as the $\overrightarrow{MT}$ vector norm.

$$Gm = \left|\overrightarrow{MT}\right| = \sqrt[2]{CpT^2 + CohT^2 + CxT^2 + WsicT^2 + (100 - SsT)^2} \quad (25)$$

This mathematical expression allowed us to determine how good or bad is the decomposition. A small *Gm* implies a good granularity. The aim is to obtain a solution with low complexity *(CxT)*, low coupling *(CpT)*, low lack of cohesion grade *(CohT)*, small *WsicT*, and high semantic similarity (*SsT* was a number between 0 and 100 so that we can minimize *Gm*, we include in the $\overrightarrow{MT}$ vector the value of 100 minus *SsT*, so values close to zero correspond to a greater semantic similarity). We tested different combinations of *CpT*, *CohT*, *CxT*, *WsicT*, and *SsT* in the *Gm* metrics, we selected the best results, and they are presented in section V.

B. PARAMETERIZE COMPONENT

It is responsible for taking input data and converting it into a format that can be processed by the grouper. It extracts the key data, such as identifier, name, description, estimated points, estimated time, scenario, observations, and dependencies, from the user story. Later, with this data, the model can group the user stories in microservices and calculate the metrics from the user stories dependencies. The format of the user stories is a CVS file where the key data (i.e., identifier, name, description, estimated points, estimated time, scenario, and observations) are supplied.

A user story describes the functionality that will be provide value to a user or customer of the software system[53], [54]. The information that a user story can contain according to Kent Beck is: the date, the type of activity (new, correction, improvement), functional test, story number or identifier, technical and customer priority, reference to another story, risk, technical estimate (points and hours), a description, notes and a follow-up list with the date, status of things to be completed and comments [55].

User (architect or development team) creates the project and loads the information of user stories from the CVS file to *MB*. Then the user defines dependencies among user stories *(HU)* according to the business logic, dataflow, database, or calls. We defined a dependence among HU_i and HU_j when HU_i calls or executes HU_j.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

C. GROUPER COMPONENT

This component groups user histories into microservices. User or architect can add up and generate automatic decompositions of these user stories in microservices (using a genetic algorithm, or a semantic grouping algorithm), or creating the decomposition manually by themselves. The semantic grouping algorithm will be addressed in future work.

1. GENETIC PROGRAMMING

The genetic algorithm seeks to find the best combination, the best assignation of stories to microservices in such a way that *Gm* is lower, using (25).

The genetic algorithms were established by Holland [56], which is iterative, in each iteration, the best individuals are selected, everyone has a chromosome, which is crossed with another individual to generate the new population (reproduction), some mutations are generated to find the optimal solution to the problem [57]. Our genetic algorithm consisted of distributing or assigning user stories to microservices automatically, considering coupling, cohesion, granularity, complexity, and semantic similarity metrics. We designed the genetic algorithm as follows. See figure 3.

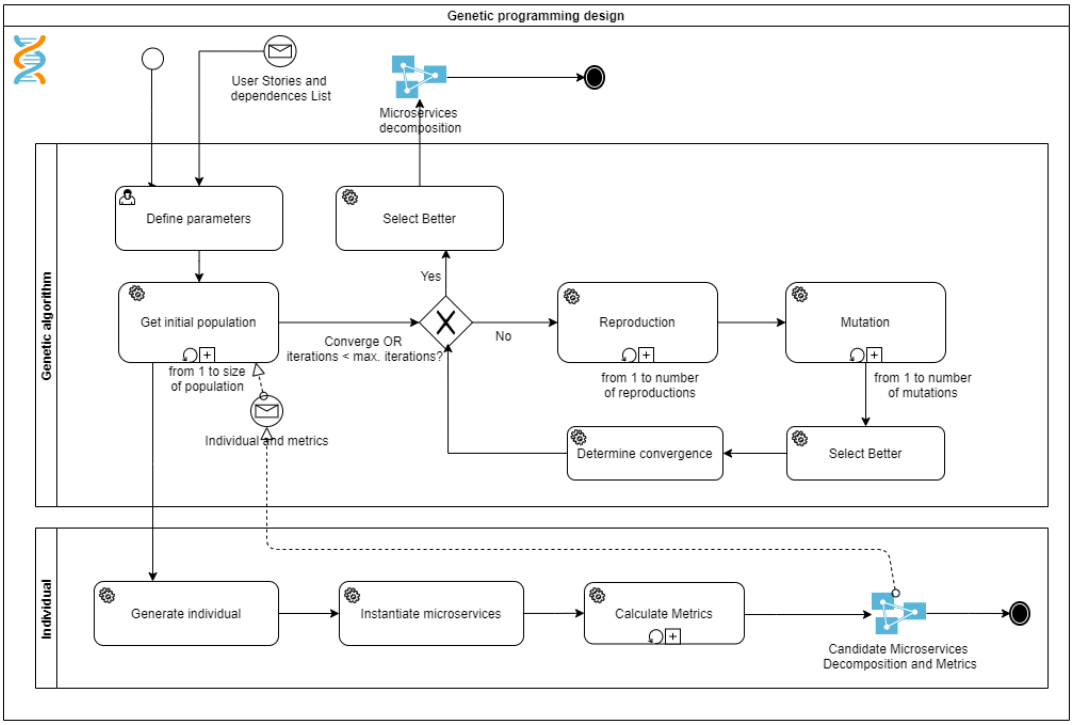


FIGURE 4. Genetic algorithm design of the Microservices backlog.

Get Initial Population Method. There is a set of user stories $HU = \{hu_1, hu_2, hu_3, ..., hu_m\}$, which must be assigned to the microservices. We have a set of microservices $MS = \{ms_1, ms_2, ms_3, ..., ms_n\}$ and some metrics calculated from the information contained in the user story. Individuals are defined from the assignment of stories to microservices; therefore, the chromosome of everyone is defined from an assignment matrix of ones and zeros, wherein the columns there are user stories and in the rows are the microservices, and the cross contains a 1 when the user story is assigned to the microservice or zero if not. In table II, an example is presented for two microservices $MS = \{ms_1, ms_2\}$ and 5 user stories $HU = \{hu_1, hu_2, hu_3, hu_4, hu_5\}$.

The resulting chromosome would be the union of the assignments of each user story to each microservice (rows), for this case, it would be:

Chromosome: 10011 01100.

From this chromosome, it was possible to define the function of adaptation or objective function, using (24).

TABLE II
EXAMPLE OF AN ASSIGNMENT MATRIX

Microservices	hu_1	hu_2	hu_3	hu_4	hu_5
ms_1	1	0	0	1	1
ms_2	0	1	1	0	0

From this chromosome, we define the adaptation function or objective function, which is based on equation (25), uses a combination of the metrics of coupling (*CpT*, equation 11), cohesion (*CohT*, equation 14), granularity (*WsicT*, equation 18), complexity (*CxT*, equation 22) and semantic similarity (*SsT*, equation 17). The objective functions used are detailed below:

$$F1 = \sqrt[2]{(10\ Cpt)^2 + CxT^2 + WsicT^2 + (100 - SsT)^2} \quad (26)$$

$$F2 = \sqrt[2]{(10\ Cpt)^2 + WsicT^2 + (100 - SsT)^2} \quad (27)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

$$F3 = \sqrt[2]{CxT^2 + (100 - SsT)^2} \quad (28)$$
$$F4 = \sqrt[2]{(10\,Cpt)^2 + CohT^2 + (100 - SsT)^2} \quad (29)$$
$$F5 = \sqrt[2]{(10\,Cpt)^2 + (100 - SsT)^2} \quad (30)$$
$$F6 = \sqrt[2]{(10\,Cpt)^2 + CohT^2 + WsicT^2 + (100 - SsT)^2} \quad (31)$$
$$F7 = \sqrt[2]{(10\,Cpt)^2 + CxT^2 + (100 - SsT)^2} \quad (32)$$
$$F8 = \sqrt[2]{(10\,Cpt)^2 + CohT^2 + WsicT^2} \quad (33)$$

Reproduction Method. A different assignment would be generated from selected parents. In this case, the father and mother are randomly selected from the population; to generate the child information is taken from the father and mother, from the assignment matrix the first columns of the father are taken, and the last columns of the mother are joined, generating a new assignment. It must be considered that a user story cannot be assigned twice, this means that in the assignment matrix only one can appear in each column. Example: Given the two chromosomes:

1) Father: 10011 01100.
2) Mother 01000 10111.

The son would be 10000 01111.

Mutation Method. The mutation indicates changing a random bit of the chromosome, changing a bit of the chromosome of this problem from 1 to 0 or from 0 to 1, implies that a user story is assigned or unassigned to a microservice and this must be assigned or unassigned to another microservice. This implies that the mutation is done on two bits. Example:

Mutate bit 7 of the obtained chromosome: 01011 10100.

Mutated chromosome: 00011 11100.

In this case, bit 7 which is zero must be changed to one, i.e. the user story in column 2 of the matrix must be assigned to the second microservice and at the same time be unassigned from the first microservice.

The mutated chromosomes must be included in the population. This process is carried out randomly, the individuals to be mutated are selected from the population, the mutation of a bit is also carried out randomly, for the mutation the value of the target function is calculated and included in the population.

Select Better Method: In the processes of genetic selection, the strongest survive, in the case of the problem of the automatic generation of the assignment of user histories to microservices, the n individuals who best adapt to the conditions of the problem survive. The assignments that imply a lower *Gm*.

The selection was from the objective function, it was applied to each individual and the population was ordered in ascending form, considering the first places, the best individuals, corresponding to the assignments involving lower *Gm* using (21).

Convergence: To determine the convergence of the method, the number of iterations or generations of the population to be processed was defined, we defined the convergence when 10% of the population converge to the same *Gm* value. If did not converge, at the end of the iterations, the algorithm is stopped, and the chromosome located in the first place was selected, which would be the best assignment of user stories to microservices. For the case studies used to evaluate the proposed method, a population of 1000 individuals was generated, with a maximum of 400 iterations or generations, with 500 children and 500 mutations in each generation. The algorithm was tested several times obtaining the same result, even with more individuals and more iterations.

D. METRICS CALCULATOR COMPONENT

The system through the metric calculator component calculates the metrics of coupling, cohesion, complexity, granularity, estimated performance (microservices requests and calls), and estimated development time. With these metrics, we can evaluate and compare the decompositions of the project to make decisions at design time. These metrics were defined in subsection A of the Microservices Backlog model.

We implemented algorithms to calculate the metrics and generate comparative tables for analyzing the microservices-based applications.

E. MICROSERVICES BACKLOG DIAGRAM AND DECOMPOSITION EVALUATOR.

Figure 4 shows Microservices Backlog for the Cargo Tracking application. The outputs of the model are the microservices backlog diagram and the metrics.

The diagram shows key information to the designer such as the size of each microservice, its complexity, dependencies, coupling, cohesion, and development time. The architect can notice at first sight that the purple microservice – Localization (see the diagram of figure 4) is a critical point of the system, because that it is massively used by all the others, if this microservice failure, then the whole system can fail. The architect at design time can already think about fault tolerance mechanisms, load balancing, and monitoring on that critical microservice. They can have a vision of the global system at design time.

The microservices backlog in figure 5 was obtained by decomposition using DDD and the following macro-algorithm:

1) Identify and describe the user stories of the application (Cargo Tracking in this case).
2) Define the dependencies among the user stories.
3) Identify the entities.
4) Define the aggregates,
5) Establish the delimited contexts and link the entities and their respective user stories.
6) Calculate metrics for each microservice and the whole application through the metric calculator component.

We highlight that the grouper component of MB automatically identifies the candidate microservices when

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

using the genetic algorithm or the grouping algorithm, then steps 3 to 5 are automatic.

After obtaining the decompositions, we can perform join or decompose operations of the microservices, perform a comparative analysis of the decompositions, and select the best one. Figure 6 presents the comparative table of a project registered in the system. Metrics are presented for each decomposition, which can be automatically ordered for analysis and comparison.

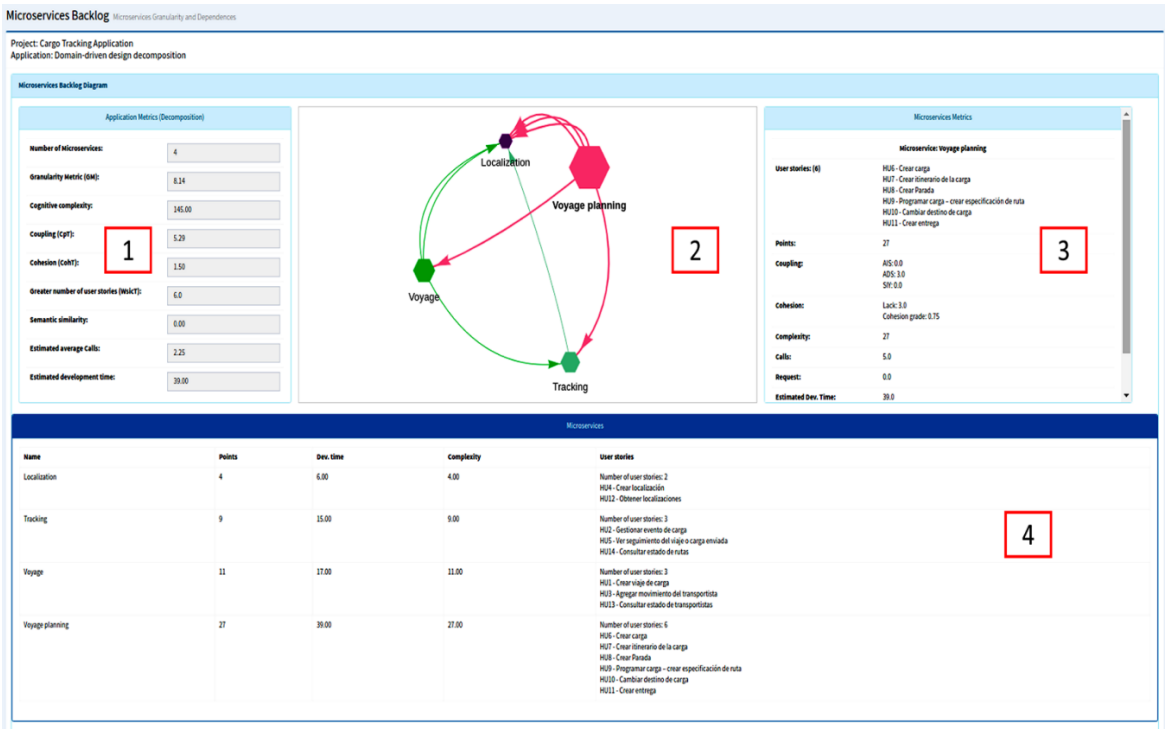
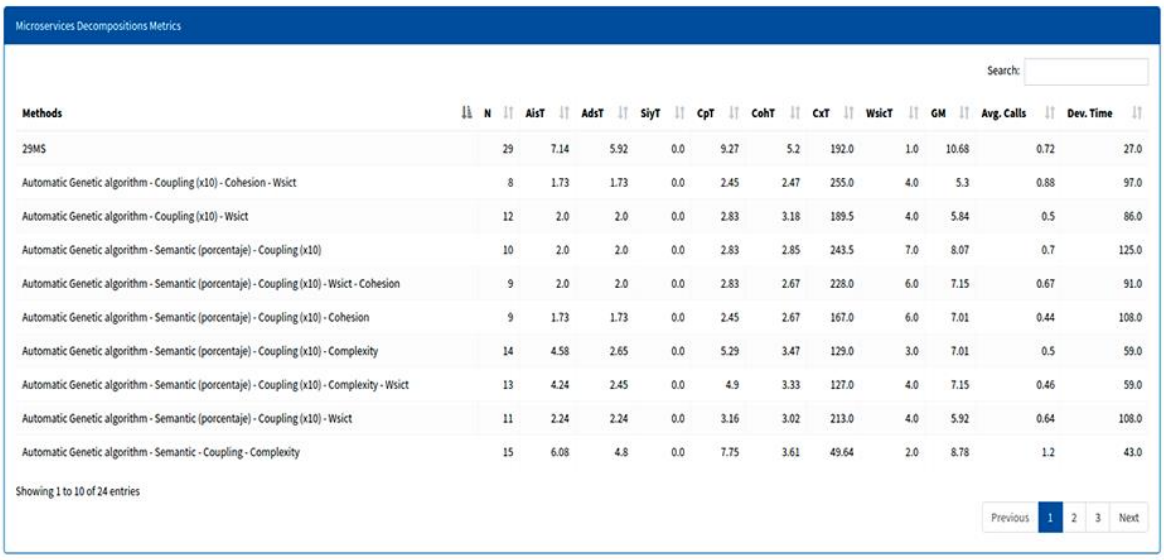


FIGURE 5. Microservices backlog for Cargo Tracking application, microservices identified using Domain-driven design. 1) MSBA metrics; 2) Dependences graph of MSBA; 3) Microservice metrics; 4) Microservices details.



Methods	N	AisT	AdsT	SiyT	CpT	CohT	CxT	WsicT	GM	Avg. Calls	Dev. Time
29MS	29	7.14	5.92	0.0	9.27	5.2	192.0	1.0	10.68	0.72	27.0
Automatic Genetic algorithm - Coupling (x10) - Cohesion - Wsict	8	1.73	1.73	0.0	2.45	2.47	255.0	4.0	5.3	0.88	97.0
Automatic Genetic algorithm - Coupling (x10) - Wsict	12	2.0	2.0	0.0	2.83	3.18	189.5	4.0	5.84	0.5	86.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10)	10	2.0	2.0	0.0	2.83	2.85	243.5	7.0	8.07	0.7	125.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10) - Wsict - Cohesion	9	2.0	2.0	0.0	2.83	2.67	228.0	6.0	7.15	0.67	91.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10) - Cohesion	9	1.73	1.73	0.0	2.45	2.67	167.0	6.0	7.01	0.44	108.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10) - Complexity	14	4.58	2.65	0.0	5.29	3.47	129.0	3.0	7.01	0.5	59.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10) - Complexity - Wsict	13	4.24	2.45	0.0	4.9	3.33	127.0	4.0	7.15	0.46	59.0
Automatic Genetic algorithm - Semantic (porcentaje) - Coupling (x10) - Wsict	11	2.24	2.24	0.0	3.16	3.02	213.0	4.0	5.92	0.64	108.0
Automatic Genetic algorithm - Semantic - Coupling - Complexity	15	6.08	4.8	0.0	7.75	3.61	49.64	2.0	8.78	1.2	43.0

FIGURE 6. Evaluate decompositions in the Microservices backlog.

V. RESULTS

We evaluated MB by comparing it with two state-of-the-art examples: Cargo Tracking and JPet Store; and one real-life project: Foristom Conferences. We compared Cargo Tracking and Jpet Store against the decomposition (microservices and their user stories) obtained with DDD, state-of-the-art approaches, and our model. Whereas the real-life project were compared against DDD, and the decomposition obtained by our model. We run the genetic algorithm for all tests with a population of 1000 individuals, convergence 10%, the maximum number of iterations 400 with 500 children and 500

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

mutations in each iteration. The following process was carried out for each case study:

1. The case study is described.
2. User stories are identified, and the product backlog is defined.
3. Dependencies between user stories are identified.
4. Decompositions are obtained for each comparison method.
5. Metrics are calculated through the metrics calculator component.
6. The solutions proposed by each method are presented graphically.
7. The metrics data table is presented for comparative analysis.
8. Comparative charts of the metrics for each method are presented.
9. The value obtained in the cognitive complexity and in the *Gm* granularity metrics is compared.
10. The best results of the genetic algorithm are compared against *DDD*.

A. CARGO TRACKING APPLICATION

In Baresi et al. [47] Cargo Tracking application is described, as follows, "the focus of the application is to move a Cargo (identified by a TrackingId) between two Locations through a RouteSpecification. Once a Cargo becomes available, it is associated with one of the Itineraries (lists of CarrierMovements), selected from existing Voyages. HandlingEvents then trace the progress of the Cargo on the Itinerary. The Delivery of a Cargo informs about its state, estimated arrival time, and is on track". We extracted and defined the user stories; the product backlog is detailed in Table II. The points and times are input data to our model.

TABLE II
PRODUCT BACKLOG FOR CARGO TRACKING APPLICATION

ID	Name	Points	Dev. Time (hours)
hu_1	Create voyage	3	5
hu_2	Handle Cargo Event	3	5
hu_3	Add Carrier Movement	5	7
hu_4	Create Location	2	3
hu_5	View Tracking.	3	5
hu_6	Create Cargo	7	10
hu_7	Route Cargo	5	7
hu_8	Create Leg	2	3
hu_9	Book cargo	5	7
hu_{10}	Change Cargo Destination	1	2
hu_{11}	Create Delivery	7	10
hu_{12}	Get Locations	2	3
hu_{13}	Get carrier status	3	5
hu_{14}	Get routes status	3	5
	Total:	51	77

ID: user story identifier. Points: estimated user story points. Dev. Time: estimated development time in hours.

A critical point of our proposed method is the dependencies between user stories. They must be identified and registered in MB through the parameterizing component, which offers the functionality to define dependencies between user stories. We define a dependence between hu_i and hu_j when hu_i calls or executes hu_j. For example, to create a voyage (hu_1) we must get the locations (hu_{12}), this implies that the hu_1 has a dependence on hu_{12}. Table III presents the dependencies identified by us among the user stories. To illustrate the proposed genetic algorithm the statement of these dependencies is valid.

TABLE III
USER STORIES DEPENDENCES FOR CARGO TRACKING

User Stories	Dependences	User Stories	Dependences
hu_1	$\{hu_{12}, hu_3\}$	hu_8	$\{hu_{12}\}$
hu_2	$\{hu_{12}\}$	hu_9	$\{hu_{12}\}$
hu_3	$\{hu_{12}\}$	hu_{10}	$\{hu_{12}\}$
hu_4	$\{\}$	hu_{11}	$\{hu_6, hu_{13}, hu_{14}\}$
hu_5	$\{\}$	hu_{12}	$\{\}$
hu_6	$\{hu_7, hu_9, hu_{11}\}$	hu_{13}	$\{hu_5\}$
hu_7	$\{hu_8\}$	hu_{14}	$\{hu_5\}$

Dependencies are used to calculate the metrics, for example, to calculate the *AIS* metric of the decomposition obtained with DDD for the microservice called Localization (see figure 5). ms_1*(Voyage)* $= \{hu_1, hu_3, hu_{13}\}$, ms_2 *(Tracking)* $= \{hu_2, hu_5, hu_{14}\}$, ms_3 *(Localization)* $= \{hu_4, hu_{12}\}$, ms_4 *(Voyage Planning)* $= \{hu_6, hu_7, hu_8, hu_9, hu_{10}, hu_{11}\}$.

The metric *AIS* is the number of clients that invoke at least one operation of a microservice's interface (see (4) and (5)). Then we count the number of microservices that invoke or use hu_4 or hu_{12} from the dependencies. hu_4 is not used by any other user stories, it does not appear in any dependencies (see table III), whereas hu_{12} is used by hu_1, hu_2, hu_3, hu_8, hu_9, and hu_{10} corresponding to 3 microservices, therefore AIS = 3. Similarly, other metrics are calculated.

Figure 6 presents the microservice backlog for the decompositions generated by the Microservice Backlog model compared with DDD, MITIA, and Service Cutter for Cargo Tracking application.

We done an analysis of the objective functions (F1 to F8), which used different combinations of *CpT*, *CohT*, *WsicT*, *CxT*, and *SsT*. The best results were using *CpT*, *CxT*, *WsicT*, and *SsT*, which contained three microservices.

All evaluated methods converged to almost the same number of microservices (3 or 4 microservices). The distribution of user stories into microservices was different.

The number of microservices was 3 or 4 in all proposed decompositions, the genetic algorithm has fewer microservices than DDD and MITIA, in the first and third method.

Our genetic algorithm obtained coupling values of 3.16, 2.83, and 2, these values were smaller or equal than DDD (2.83), Service Cutter (3.16), and MITIA (6.78); the smaller coupling was the genetic algorithm using *CpT*, *CohT*, and *WsicT* in the objective function, therefore the decomposition obtained by MB has low coupling (see table IV, we highlighted the solution with lower *Gm*).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

The cohesion values were 1.16, 1.5, and 1.16 for the genetic algorithm, whereas DDD (1.5), Service Cutter (1.15), and MITIA (1.06) obtained similar values; the semantic similarity of the genetic algorithm was greater than 70%, so our genetic algorithm obtained coherent decompositions from the semantic point of view, therefore the semantic cohesion was high. The smaller number of calls among microservices correspond to the genetic algorithm and MITIA (5 calls), whereas DDD (6 calls) and Service Cutter (10 calls) had more calls; then the genetic algorithm obtained solutions with fewer dependencies and less communication among microservices.

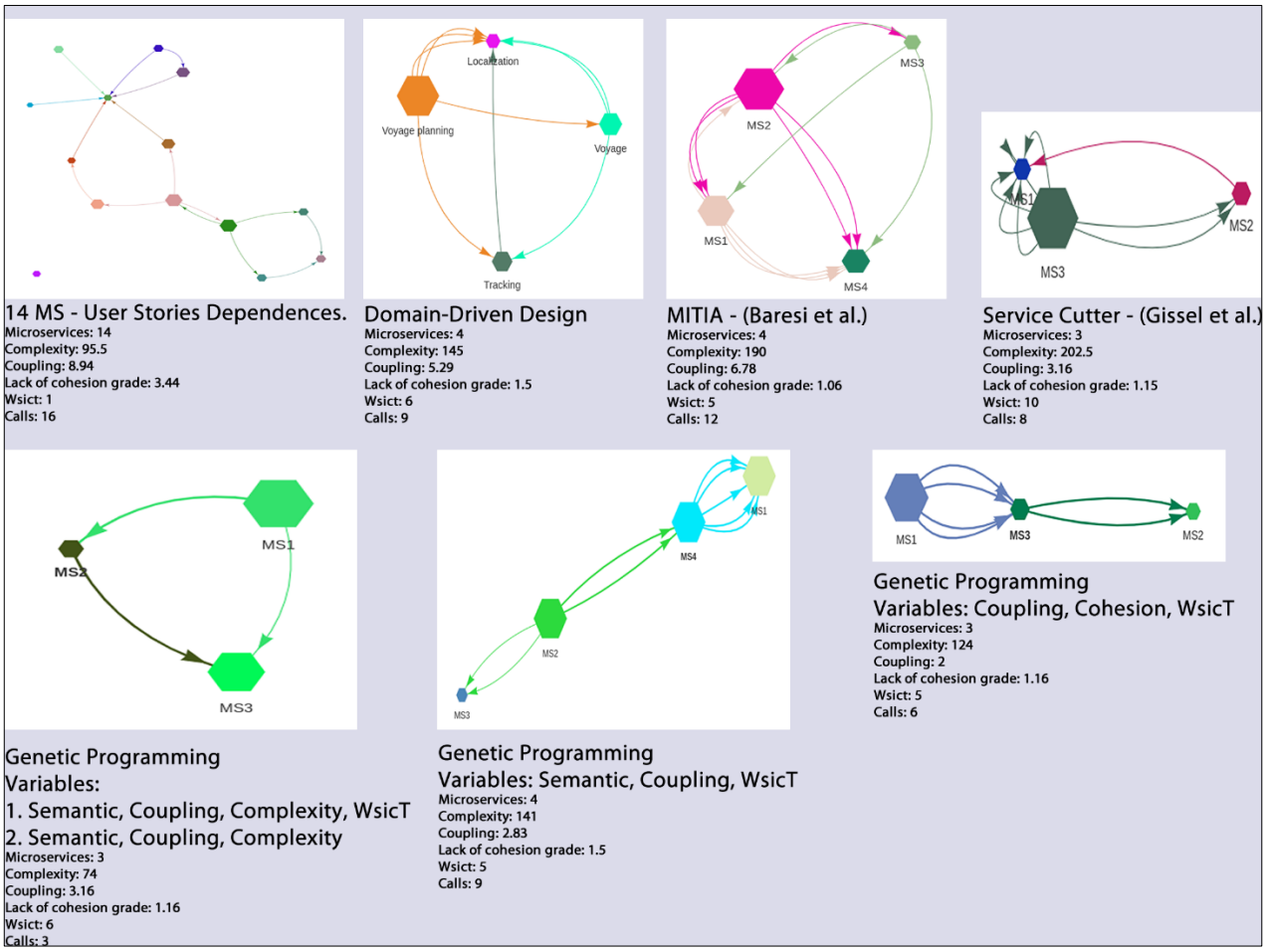


FIGURE 7. Microservice Backlog model compared with DDD, MITIA, and Service Cutter for Cargo Tracking application.

TABLE IV

COMPARATIVE ANALYSIS OF DECOMPOSITIONS FOR CARGO TRACKING APPLICATION

Method / Metrics	Coupling				Cohesion		Granularity		Perfor.	Complexity		T	GM
	AisT	AdsT	SiyT	CpT	CohT	SsT	N	WsicT	Calls	Max. P_i	CxT		
MB - Genetic algorithm F1: CpT, CxT, WsicT, SsT F7: CpT, CxT, SsT	**2.24**	**2.24**	**0**	**3.16**	**1.16**	**70.9**	**3**	**6**	**3**	**23**	**74.0**	**35**	**85.8**
MB - Genetic algorithm F2: CpT, WsicT, SsT	1.73	2.24	0	2.83	1.50	75.0	4	5	9	14	141.0	21	146.1
MB - Genetic algorithm F8: CpT, CohT, WsicT	1.41	1.41	0	2.00	1.16	70.8	3	5	6	22	124.0	33	129.1
DDD	3.74	3.74	0	5.29	1.50	74.1	4	6	9	27	145.0	39	156.6
Service Cutter	2.24	2.24	0	3.16	1.15	74.4	3	10	8	41	202.5	61	206.8
MITIA	4.24	4.69	2.45	6.78	1.06	76.8	4	5	12	19	190.0	30	203.1
14MS (Finer granularity)	6.93	5.48	1.41	8.94	3.44	100.0	14	1	16	7	95.5	10	130.9
Monolith (Greater granularity)	0	0	0	0	0	69.2	1	14	0	51	32.5	77	46.9

The decomposition performed by our method was different from DDD, our model did not group the entities and their stories or operation that make up the aggregate, neither considered transactions among user stories or business logic of the application. These topics will be considered in future work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

In the decomposition obtained with the genetic algorithm, the critical point of failure of the proposed DDD solution is removed, Localization microservice is used for all microservices. The number of calls between microservices is reduced, thus improving performance. The maximum number of operations associated with a microservice is also reduced, as well as the cognitive complexity and the estimated development time. In the decomposition generated by genetic programming, more microservices can work independently without depending on other microservices. Whereas in the solution proposed by DDD, only one microservice can work independently. In the decomposition proposed by DDD, there are more dependencies.

By distributing user stories differently, shorter development times of the entire system can be obtained. Considering that each microservice is developed by an independent team in parallel.

Figure 8 shows graphically the comparative analysis of the evaluation metrics. Figure 9 shows specifically the cognitive complexity obtained by the methods studied.

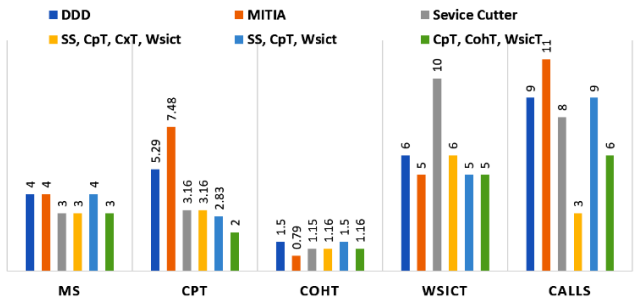


FIGURE 8. Comparative analysis of evaluation metrics of Cargo Tracking application. MS: Number of microservices, CPT: Coupling of MSBA, COHT: Lack of cohesion grade of MSBA, WSICT: Maximum WSIC of MSBA, WSIC is the number of user stories of the MS. CALLS: number of calls between microservices.

The number of calls of our approach is less than DDD, Service Cutter, and MITIA. This metric measure or determine the degree of dependence that have the microservices that are part of the application, a larger value implies a greater dependence and lower performance because they require the execution of operations that belong to other microservices in other containers.

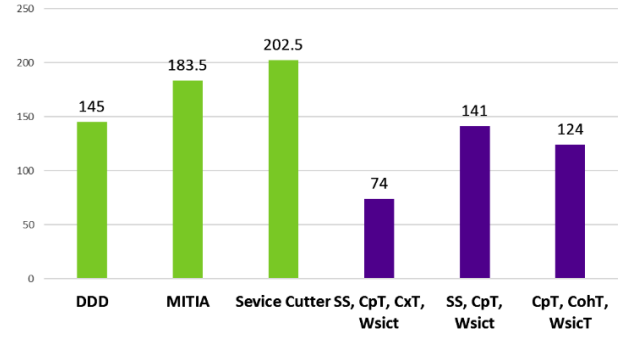


FIGURE 9. Comparative analysis of cognitive complexity points of Cargo Tracking application.

Cognitive complexity metrics estimate the difficulty of understanding, implementing, and maintaining the microservice-based application, depending on the complexity of each microservice, the interactions between them, and their number. MB obtained (74, 141, and 124) lower cognitive complexity points than DDD (145), MITIA (190), and Service Cutter (202.5); therefore, MB can reduce the complexity of the microservice-based applications.

The results obtained for semantic similarity are very similar in all proposed decompositions. It can be highlighted that the semantic similarity for all cases exceeds 70%, therefore, the decompositions were coherent from the semantic point of view. Figure 10 shows the comparative analysis of the metric *GM*. The lower *Gm* value corresponds to the genetic algorithm (85.79) of MB, whereas DDD (156.64), MITIA (203.14), and Service Cutter (206.79), Therefore, the decomposition obtained by MB corresponds to the best solution to the problem. We observed that the solution proposed with less complexity, less coupling, and fewer calls also corresponds to the one with the lowest value of the *Gm* metric and corresponds to the genetic algorithm of MB.

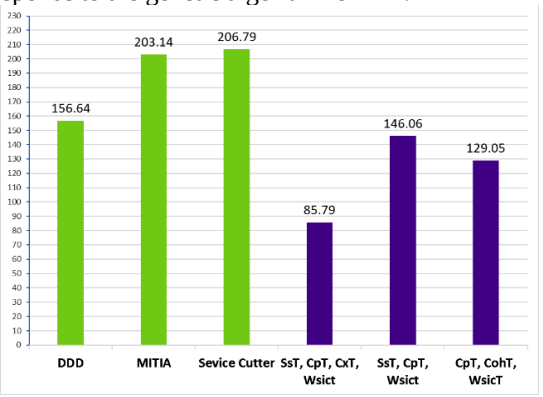


FIGURE 10. Comparative analysis of granularity metrics of Cargo Tracking application.

Finally figure 11 presents the comparative results for Cargo Tracking, comparing MB genetic algorithm against DDD.

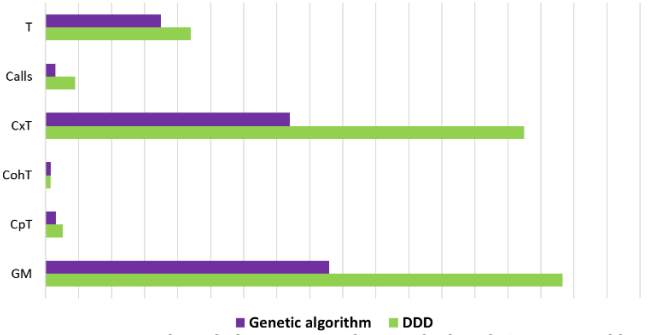


FIGURE 11. Results of the comparative analysis of Cargo Tracking application. T: Estimated development time.

Where T is the estimated development time of the decomposition obtained by each method. The genetic algorithm obtains less estimated development time, fewer calls between microservices, less complexity, greater cohesion, less coupling, and less *Gm* than DDD.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

B. JPET STORE APPLICATION

The JPetStore Demo[1] is an online pet store. We can browse and search the product catalog, choose items to add to a shopping cart, amend the shopping cart, and order the items in the shopping cart. You can perform many of these actions without registering with or logging into the application. However, before you can order items you must log in (sign in) to the application. To sign in, you must have an account with the application, which is created when you register (sign up) with the application [58]. The functionalities of Jpet Store are listed below.

- Signing Up
- Signing In
- Working with the Product Catalog: Browsing the Catalog, Searching the Catalog
- Working with the Shopping Cart: Adding and Removing Items, Updating the Quantity of an Item, Ordering Items, Reviewing an Order

This application has been used to validate decomposition methods and migrations from monolithic applications to microservices [18], [29], [35]. From the description, definition, and source code of the application, the following user stories were identified. We identified the operations; from them, we specify and estimate the user stories (see Table V).

TABLE V
PRODUCT BACKLOG FOR JPET STORE APPLICATION

ID.	Name	Points	Dev. Time (Hours)
hu_1	View category	3	5
hu_2	List categories	1	3
hu_3	Search products	5	7
hu_4	View product	3	5
hu_5	View item (get Item)	3	5
hu_6	Add item to cart	5	7
hu_7	Remove item from cart	3	5
hu_8	Update cart quantities	3	5
hu_9	Get cart	3	5
hu_{10}	New order	7	10
hu_{11}	Get order	3	5
hu_{12}	Set order ID	5	7
hu_{13}	List orders	3	5
hu_{14}	Is authenticated	3	5
hu_{15}	New account (Sign up)	5	7
hu_{16}	Get account	3	5
hu_{17}	Sign off	2	3
hu_{18}	Update account	3	5
hu_{19}	getCategory	2	3
hu_{20}	getProduct	2	3
hu_{21}	Is Item in Stock	3	5
hu_{22}	Sign in	3	5
	Total	73	115

The points and times were estimated according to our experience and correspond to the effort and time it would take for us to develop each user story; the total time corresponds to a sequential order of development of the user stories. In real-life software development using agile methodologies, this estimate would be made by the development team in the release planning considering their own characteristics and speed of development.

A dependency is defined when a user story uses or calls another user story. This example can be considered as migration from monolith to microservices, in this case, the user stories can be replaced by the operations/methods or services of the application; a dependency corresponds to an execution dependency, in which an operation calls another operation to fulfill its purpose.

In this example, the source code of the monolithic application was available. To define the dependencies among user stories, the source code was analyzed to identify invocation dependencies between user stories and/or operations (OrderService, CatalogService, AcountService, Cart entity, and other entities). The process is detailed below:

Dependencies of hu_1 - View category:

ViewCategory method calls to *ListCategories* method. It corresponds to another user story *(hu_2)*.

ViewCategory calls to *catalogService.getProductListByCategory* – It is the same user story and corresponds to its implementation.

ViewCategory calls to *catalogService.getCategory(id)* – It corresponds to another user story *(hu_{19})*,

Therefore, the dependencies of hu_1 are hu_2 and hu_{19}. $hu_1 = \{hu_2, hu_{19}\}$.

Dependences of hu_2 - Listar categorías - List categories:

ListCategories calls to accountAction.getCategories() – It is the same user story and corresponds to its implementation. Therefore, hu_2 has not dependencies. $hu_2 = \{\}$.

Dependencies of HU_6 - Add item to cart:

AddItemtoCart calls to *cart.incrementQuantityByItemId(workingItemId)*. It is the same user story and corresponds to its implementation.

AddItemtoCart calls to *catalogService.isItemInStock(workingItemId)*. It corresponds to another user story *(hu_{21})*

AddItemtoCart calls to catalogService.getItem(workingItemId); It corresponds to another user story *(hu_5)*.

Therefore, HU_6 has two dependencies HU_{21} and HU_5. $HU6 = \{HU_5, HU_{21}\}$.

TABLE VI
USER STORIES DEPENDENCIES FOR JPET STORE APPLICATION

Id.	Dependencies	Id.	Dependencies
hu_1	$\{hu_2, hu_{19}\}$	hu_{12}	$\{\}$
hu_2	$\{\}$	hu_{13}	$\{hu_{16}\}$
hu_3	$\{\}$	hu_{14}	$\{\}$
hu_4	$\{hu_{20}\}$	hu_{15}	$\{hu_{16}, hu_1\}$
hu_5	$\{\}$	hu_{16}	$\{\}$
hu_6	$\{hu_5, hu_{21}\}$	hu_{17}	$\{\}$
hu_7	$\{\}$	hu_{18}	$\{hu_{16}, hu_1\}$
hu_8	$\{\}$	hu_{19}	$\{\}$
hu_9	$\{\}$	hu_{20}	$\{\}$
hu_{10}	$\{hu_9, hu_{16}\}$	hu_{21}	$\{\}$
hu_{11}	$\{hu_{16}\}$	hu_{22}	$\{hu_1\}$

[1] http://demo.kieker-monitoring.net/jpetstore/help.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

We did this process for the other user stories. Table VI shows the identified dependencies.

We compared the decompositions of user stories to microservices obtained by our model against DDD, and Execution Traces, see figure 12. The results obtained by DDD and Execution traces were the same, only one user story was in a different microservice, but the metrics and comparison were equals.

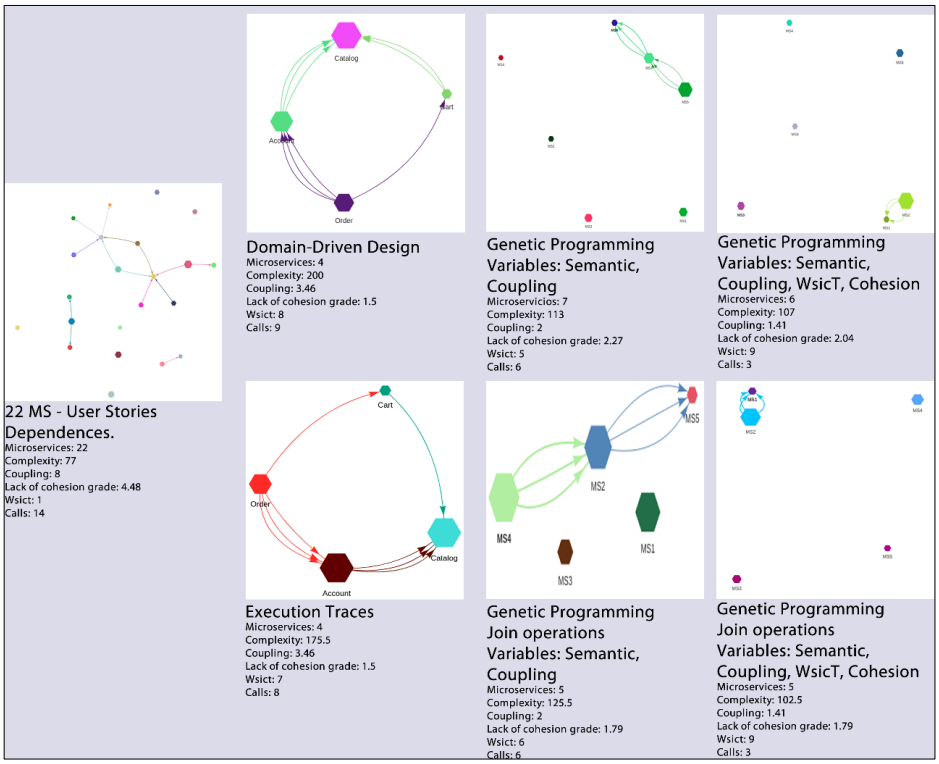


FIGURE 12. Microservice Backlog model compared with DDD and Execution Traces for JPet Store application.

TABLE VII

COMPARATIVE ANALYSIS OF DECOMPOSITIONS FOR JPET STORE APPLICATION

Method / Metrics	Coupling				Cohesion		Granularity		Perfor.	Complexity		T	Gm
	AisT	AdsT	SiyT	CpT	CohT	SsT	N	WsicT	Calls	Max P_i	CxT		
MB - Genetic algorithm CpT, SsT	1.41	1.41	0	2.00	2.27	89.4	7	5	6	21	113.0	32	115.4
MB - Genetic algorithm and Joins CpT, SsT	1.41	1.41	0	2.00	1.79	85.9	5	6	6	21	125.5	32	128.2
MB - Genetic algorithm CpT, CohT, WsicT, SsT	1	1	0	1.41	2.04	88.1	6	9	3	35	107.0	54	109.0
MB - Genetic algorithm and Joins CpT, CohT, WsicT, SsT	**1**	**1**	**0**	**1.41**	**1.79**	**86.5**	**5**	**9**	**3**	**35**	**102.5**	**54**	**104.7**
DDD	2.45	2.45	0	3.46	1.50	85.3	4	8	9	22	200.0	36	203.7
Execution Traces	2.45	2.45	0	3.46	1.50	84.1	4	7	8	19	175.5	31	179.7
22MS (Finer granularity)	6.32	4.9	0	8.00	4.48	100.0	22	2	14	7	77	10	111.1
Monolith (Greater granularity)	0	0	0	0	0	70.82	1	22	0	73	47.5	115	22.0

Automatically MB obtained more candidate microservices (seven and six microservices) than DDD (four microservices) and Execution traces (four microservices), the user did some join operations of microservices and got five microservices, this solution was close to DDD and Execution traces.

The decompositions proposed by our model had semantic similarity coherence (greater than 85%) and maybe a good candidate solution to the problem. The comparative analysis of the metrics is detailed in table VII.

The decomposition which had more coupling was 22MS (the finer granularity), if we followed the single responsibility principle, then we may associate one user story with only one microservice, thus this may increase the coupling of global application; therefore the single responsibility principle my be "group things that referred to the same things", so the semantic similarity is fundamental for grouping similar things. Our model grouped the user stories the referred to the same entity keeping low coupling and high cohesion.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

MB obtained the lowest coupling (1.41) for this application than DDD and "Execution Traces" (3.46); similar lack of cohesion (1.79) than DDD and "Execution Traces" (1.5), the lowest WsicT (5), and the smaller complexity (102.5) compared to DDD (200), and "Execution Traces" (175.5); MB presented a smaller number of calls (3 calls) among microservices than DDD (9 calls) and "Execution Traces" (8 calls); the estimated development time of the solution proposed by our model (32 hours) was lower than DDD (36 hours) and close to "Execution Traces" (31 hours), as well as the maximum number of user story points associated with a microservice (MB 21 points, DDD 22 points, and "Execution Traces" 19 points).

Changing a user story or operation from one microservice to another can have consequences on performance, complexity, and coupling; therefore it is an important point to consider when designing microservice-based applications and should be done based on metrics such as those proposed in this paper, where the impact of those changes and different distributions of user stories or operations on microservices can be graphically analyzed, all at design time.

Figure 13 shows the comparative analysis of the metrics and figure 14 shows the complexity of the obtained decompositions of JPet Store. The first two bars correspond to DDD and "Execution traces", the others correspond to MB.

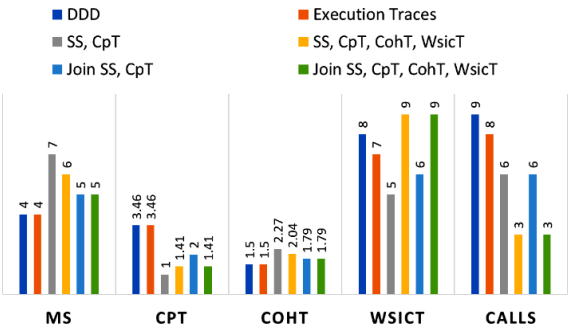


FIGURE 13. Comparative analysis of evaluation metrics of Jpet Store. MS: Number of microservices, CPT: Coupling of MSBA, COHT: Lack of cohesion grade of MSBA, WSICT: Maximum WSIC of MSBA, WSIC is the number of user stories of the MS. CALLS: number of calls between microservices. SS: Semantic similarity.

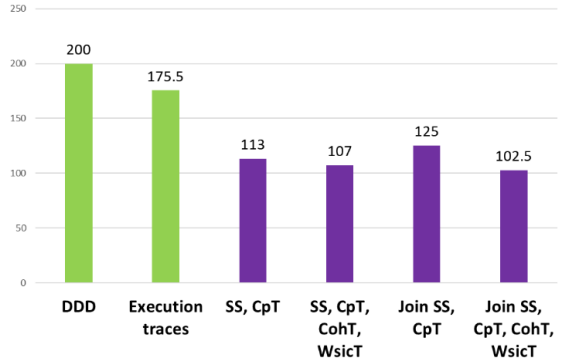


FIGURE 14. Comparative analysis of cognitive complexity points of Jpet Store application. Where SS: Semantic similarity.

Figure 15 shows the comparative analysis of the granularity metric *Gm*. We observed that the solutions proposed by the genetic algorithm obtained lower Gm (104.7) than DDD (203.7), and "Execution Traces" (179.7).

The genetic algorithm of the MB obtained greater semantic similarity (89.4%) than DDD (85.3%) and "Execution Traces" (84.1%); additionally, MB obtained a high semantic similarity value, being greater than 85% for all cases; then the MB obtained coherent decompositions from the semantic point of view, indicating a high semantic cohesion.

FIGURE 15. Comparative analysis of the granularity metric of Jpet-Store application.

The comparative analysis of Jpet-Store application is presented in figure 16, we compared the solution with lower *Gm* against DDD; where T is the estimated development time.

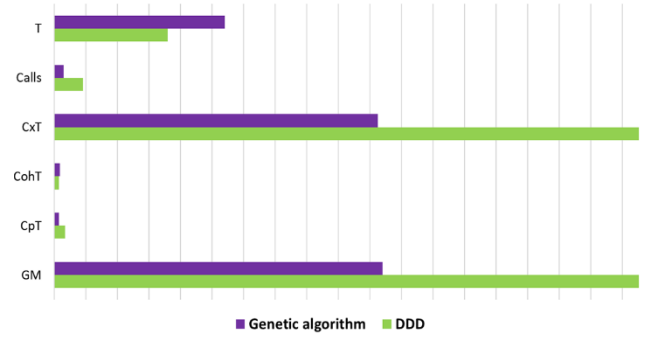


FIGURE 16. Comparative analysis of Jpet-Store application.

The genetic algorithm obtained lower coupling, higher cohesion, lower complexity, high semantic cohesion, fewer calls between microservices, and lower *Gm* compared to DDD.

MB obtained decomposition for the hypothetical projects Cargo Tracking and Jpet-Store with less coupling, less complexity, less communication between microservices (less "calls"), greater cohesion, lower *Gm* value, and shorter development time compared to DDD. This is a promising and important result because DDD is one of the most widely used methods for defining the granularity of microservices.

The results of our model were similar in JPet-Store and Cargo Tracking applications, MB obtained low coupling, low

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

or similar lack of cohesion, high semantic cohesion, small calls, and low complexity than the state-of-the-art approaches.

C. FORISTOM CONFERENCES APPLICATION

Foristom conferences is a web application that allows the management of information and organization of virtual conferences of the Foristom Foundation[2]. Foristom conferences allow us to manage everything from the creation and dissemination of the conference to the publication and presentation of the articles submitted. The Foristom Foundation is a non-profit organization. From the description and definition of the case study, the following user stories were identified, which will be implemented following the microservices architecture. The product backlog of Foristom Conferences is detailed in table IX.

In this case, the dependencies were defined according to the business logic of the application and the data flow between the different user stories. See Table X.

The aim was to compare the design proposed by the architect using DDD against the design obtained with MB. When using DDD, following the approaches proposed by Evan [46],[59], then the entities, valuable objects, delimited contexts must be identified to propose the microservices that are going to be part of the application.

Figure 17 shows the decompositions for Foristom Conferences.

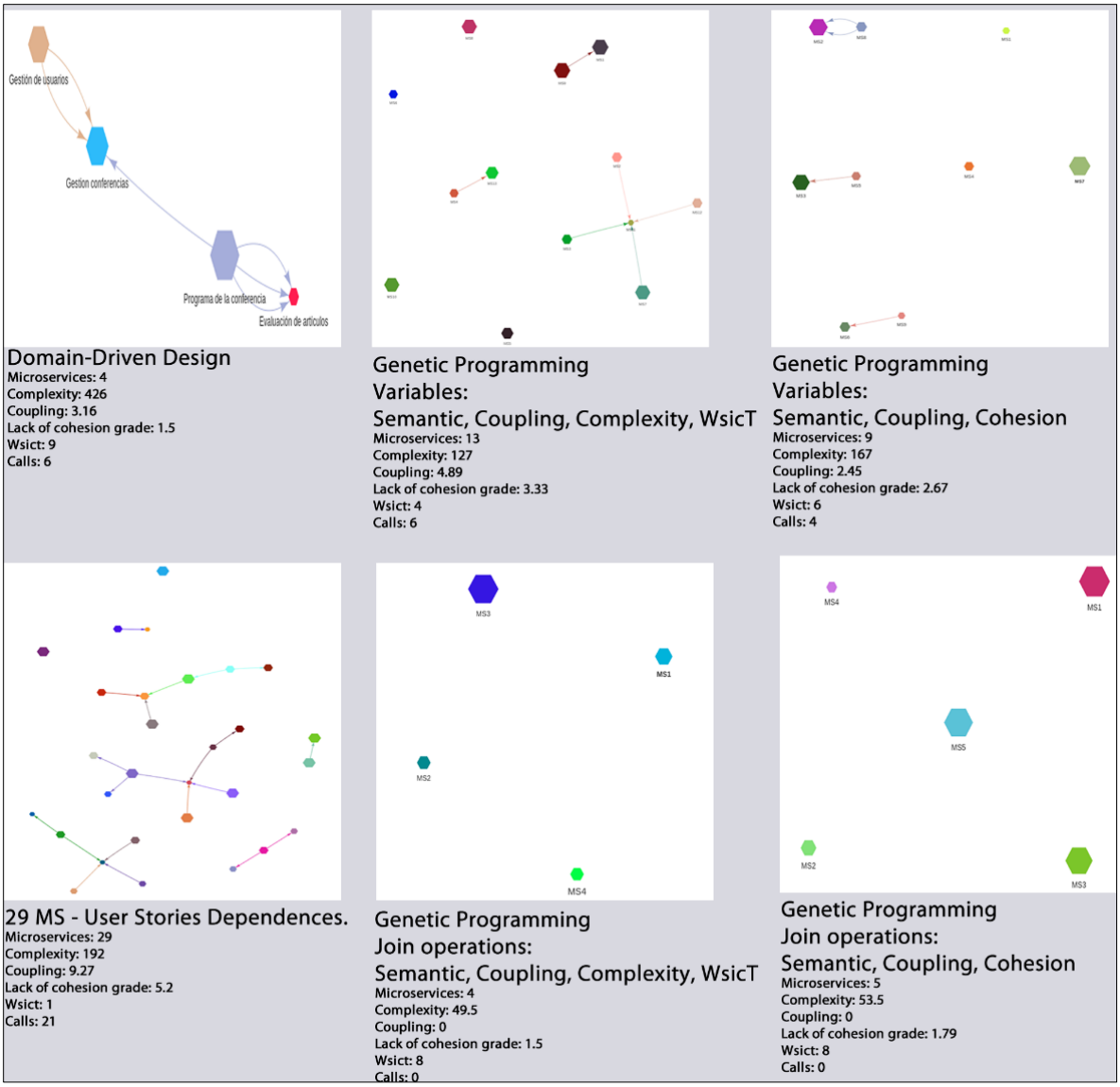


FIGURE 17. Microservice Backlog model compared with DDD for Foristom Conferences application.

[2] www.foristom.org

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

TABLE VIII

COMPARATIVE ANALYSIS OF DECOMPOSITIONS FOR FORISTOM CONFERENCES APPLICATION

Method / Metrics	Coupling				Cohesion		Granularity		Perform.	Complexity		T	Gm
	AisT	AdsT	SiyT	CpT	CohT	SsT(%)	N	WsicT	Calls	Max. Points	CxT		
Genetic algorithm CpT, CxT, WsicT, SsT	4.24	2.45	0	4.9	3.33	84.6	13	4	6	29	127	59	137.1
Genetic algorithm and Joins CpT, CxT, WsicT, SsT	**0**	**0**	**0**	**0**	**1.50**	**74.4**	**4**	**8**	**0**	**67**	**49.5**	**134**	**56.3**
Genetic algorithm CpT, CohT, SsT	1.73	1.73	0	2.45	2.67	79.5	9	6	4	52	167	108	170.2
Genetic algorithm and Joins CpT, CohT, SsT	0	0	0	0	1.79	75.3	5	8	0	67	53.5	134	59.5
DDD	2.24	2.24	0	3.16	1.50	75.7	4	9	6	83	426	167	428.0
29MS (Finer granularity)	7.14	5.92	0	9.27	5.20	100.0	29	1	21	13	192	27	213.2
Monolith (Greater granularity)	0	0	0	0	0	72.0	1	29	0	235	132	469	138.0

TABLE IX

PRODUCT BACKLOG FOR FORISTOM CONFERENCES APPLICATION

Id.	Name	Points	Dev. Time (Hours)
hu_1	Create conference.	13	27
hu_2	Get conference by id.	2	3
hu_3	Edit conference	5	9
hu_4	Delete conference	5	9
hu_5	Upload conference's support files	8	16
hu_6	Get conference's support files	2	3
hu_7	Generate conference landing page.	8	16
hu_8	Generate call for papers.	13	27
hu_9	Apply filters and specialized search on conferences.	8	16
hu_{10}	Generate conference history.	8	16
hu_{11}	Register on the conferences system.	8	16
hu_{12}	Log in the conferences system.	8	16
hu_{13}	List submitted papers by state.	2	3
hu_{14}	Submit a paper to the conference.	13	27
hu_{15}	Register the paper evaluation.	8	16
hu_{16}	Generate paper evaluation board.	5	9
hu_{17}	Register for a conference as an attendee or author	13	27
hu_{18}	Activate user (attendee or author) registration	8	16
hu_{19}	Get the list of registrations by state.	2	3
hu_{20}	Pay the conference registration fee online.	13	27
hu_{21}	Get the conferences that are taking place "On-line"	13	27
hu_{22}	Generate the home page of the conference.	13	27
hu_{23}	List conference program	5	9
hu_{24}	Create conference program, keynotes, and sessions.	13	27
hu_{25}	Enter to the meeting room as a speaker or assistant.	8	16
hu_{26}	Upload files of the presentation or session (video, photo, presentation)	5	9
hu_{27}	Generate abstract book.	13	27
hu_{28}	Download certificate of attendees and speakers.	8	16
hu_{29}	Register the presentation of a paper.	5	9
Total		235	469

For the identified microservices, complexity, coupling, granularity metrics were calculated, and development time was estimated. In this way, the architect can graphically observe several solutions or decompositions, compare them, evaluate them, and select the one he wants to implement. The comparative metric analysis of the proposed solutions to Foristom Conferences application is presented in Table VIII. The results were like Cargo Tracking and JPet-Store applications results.

TABLE X

USER STORIES DEPENDENCIES FOR FORISTOM CONFERENCES APPLICATION

Id.	Dependences	Id.	Dependences
hu_1	{}	hu_{16}	{hu_{13}, hu_{15}}
hu_2	{}	hu_{17}	{hu_{20}}
hu_3	{hu_2}	hu_{18}	{hu_{19}}
hu_4	{hu_2}	hu_{19}	{}
hu_5	{hu_2, hu_6}	hu_{20}	{}
hu_6	{}	hu_{21}	{hu_9}
hu_7	{hu_2}	hu_{22}	{hu_{23}, hu_{13}, hu_{28}}
hu_8	{hu_9}	hu_{23}	{}
hu_9	{}	hu_{24}	{hu_{13}}
hu_{10}	{hu_9}	hu_{25}	{hu_{26}, hu_{29}}
hu_{11}	{}	hu_{26}	{}
hu_{12}	{hu_{11}, hu_8}	hu_{27}	{hu_{13}}
hu_{13}	{}	hu_{28}	{}
hu_{14}	{}	hu_{29}	{}
hu_{15}	{}		

We tested different objective functions of the genetic algorithm. The best results were with *CpT*, *CxT*, *WsicT*, and *SsT* metrics, which obtained 13 microservices, the user did a few simple joins operations, so he could reduce de decomposition to 4 microservices (equal to DDD); another good solution was with *CpT*, *CohT*, *SsT* metrics in the objective function, which automatically obtained 9 microservices, next the user could reduce to 5 microservices.

The greater coupling was 29MS decomposition (the finer granularity with 9.27), the coupling of de decomposition obtained by MB was zero, this means that the microservices had not dependences and they were independent, whereas the coupling of DDD was 3.16. Our model has less *WsicT (*4 user stories, DDD had 9*)*, lower cognitive complexity (49.5, DDD had 426), fewer estimated development time (59 hours), and user story points (29 points) than DDD (167 hours and 83 points).

These results can be seen graphically in figure 18 and figure 19.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

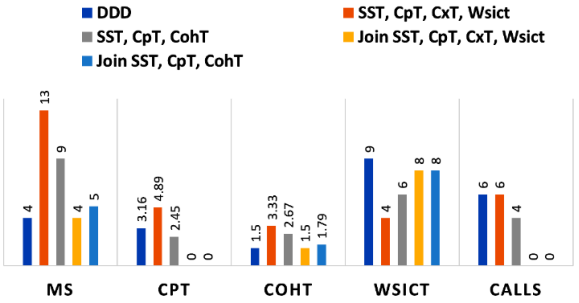


FIGURE 18. Comparative analysis of evaluation metrics of Foristom Conferences application. MS: Number of microservices (N), CPT: Coupling of MSBA, COHT: Lack of cohesion grade of MSBA, WSICT: Maximum WSIC of MSBA, WSIC is the number of user stories of the microservice. CALLS: number of calls between microservices. SST: Semantic similarity.

The cognitive complexity of MB (49.5) was considerably less than the complexity of the decomposition proposed by DDD (426). The same result was repeated as in the previous cases, so we concluded that our model obtains solutions of less complexity, thus being easier to implement and maintain.

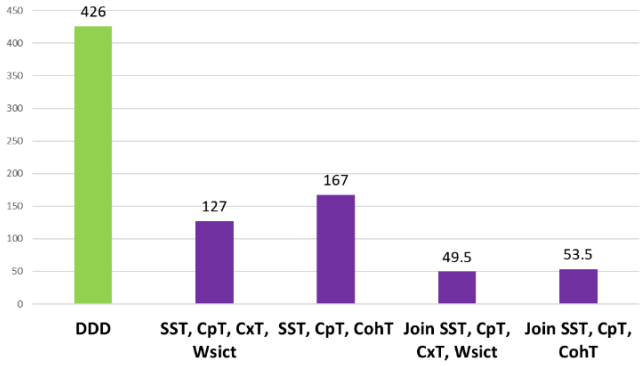


FIGURE 19. Comparative analysis of cognitive complexity points of Foristom Conferences application.

The decompositions proposed by our model were semantically and functionally coherent (greater than 74%); we were able to obtain completely independent microservices, this being an important feature to implement, maintain and deploy a microservice-based application.

If the proposed solutions present a high semantic similarity (greater than 70%), then they suggest that microservices group the stories that refer to the same entity, therefore, their cohesion is high. All obtained solutions present high semantic similarity.

We observed that it is not enough to have only high semantic similarity to consider a good distribution of user stories in microservices, other factors such as coupling, dependencies, and communication among the application microservices must be analyzed.

Figure 20 details the results of the granularity metric *Gm* for the decompositions obtained by MB compare to DDD.

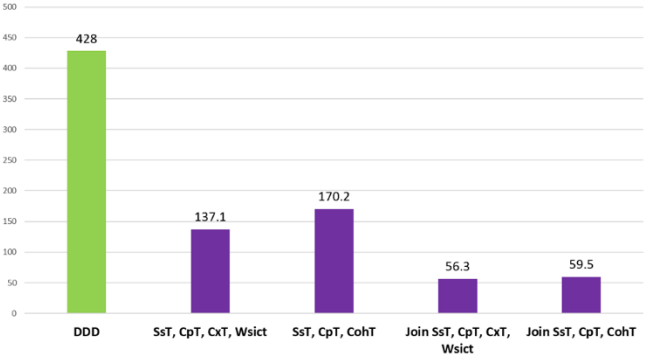


FIGURE 20. Comparative analysis of granularity metric for Foristom Conferences application.

The results of *Gm* were considerably lower for the genetic algorithm with 56.3, whereas DDD was 428. Figure 21 shows the comparative analysis for the best solutions (those with the lowest *Gm*) proposed by the genetic algorithm compared to DDD.

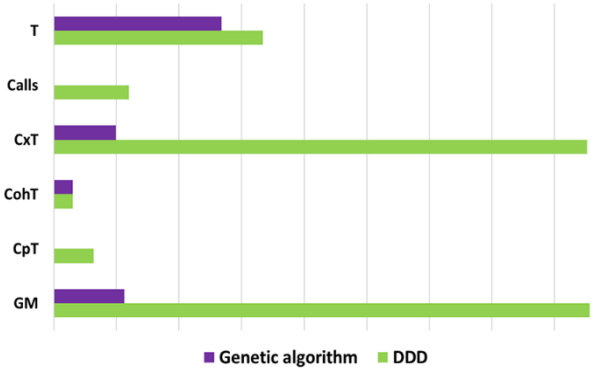


FIGURE 21. Comparative analysis of Foristom Conferences application.

The join and disjoint operations were essential to obtain better results than DDD, automatically in some cases similar or better results are obtained than DDD, but we should always check that the user stories were associated in the right place, for example, the semantic similarity algorithm assumes that the presentation session is semantically very similar to the user's session, being two different things, for this reason, the user's intervention is very important to analyze and evaluate what is obtained automatically, in order to propose improvements and get better results.

In this case study, we demonstrated that the MB obtained decompositions from user stories to microservices with low coupling, high cohesion (from the semantic point of view), low complexity, low communication between microservices, and shorter estimated development time; therefore, MB is a viable option for the design and evaluation of the granularity of microservices-based applications.

In summary, the analysis of the results obtained in this research work is presented in Table XI, comparing the results obtained with each method in all projects.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3106342, IEEE Access

We analyzed the results, the solution proposed by MB presented low coupling, high cohesion, low complexity, less communication, and fewer dependencies compared to the solution proposed by state-of-the-art methods and DDD; additionally, the proposed solutions were coherent from the semantic point of view (high semantic similarity, *SsT* greater than 70% in all cases); therefore, the proposed model MB improves the decomposition and identification of microservices.

TABLE XI

SUMMARY OF MICROSERVICES BACKLOG RESULTS

Case study	Method	Metrics								
		N	GM	CpT	CohT	SsT	WsicT	CxT	T	Calls
Cargo-Tracking	MB - Genetic algorithm	3	85.79	3.16	1.16	70.92	6	74.0	35	3
	DDD	4	156.64	5.29	1.60	74.09	6	145.0	44	9
	Service Cutter	3	206.79	3.16	1.15	74.42	10	202.5	61	8
	MITIA	4	203.14	6.78	1.06	76.77	5	190.0	30	12
Jpet-Store	MB - Genetic algorithm	5	104.75	1.41	1.79	86.50	9	102.5	54	3
	DDD	4	203.70	3.46	1.50	85.30	8	200.0	36	9
	Execution Traces	4	179.70	3.46	1.50	84.06	7	175.5	31	8
Foristom Conferences	MB - Genetic algorithm	4	56.32	0.00	1.50	74.40	8	49.5	134	0
	DDD	4	428.00	3.16	1.50	75.69	9	426.0	167	6

We analyzed the results, the solution proposed by MB presented low coupling, high cohesion, low complexity, less communication, and fewer dependencies compared to the solution proposed by state-of-the-art methods and DDD; additionally, the proposed solutions were coherent from the semantic point of view (high semantic similarity, SsT greater than 70% in all cases); therefore, the proposed model MB improves the decomposition and identification of microservices.

VI. LIMITATIONS

To determine the user stories of the state-of-the-art case studies, we used the information reported in the published papers, we studied their business logic, and we made our best effort not to bias this definition. The definition and description of the user stories were reviewed by each of the authors independently and contradictions were resolved by common agreement among the authors. We selected those state-of-the-art case studies because they were the most used in the related works.

Few datasets of microservices projects with user stories were identified, we found that Rahman et al. [60] shared a dataset composed of 20 open-source projects using specific microservice architecture patterns, and Marquez and Astudillo [61] shared a dataset of open-source microservice-based projects when investigating actual use of architectural patterns; those projects did not specify the user stories.

The definition of the dependencies among user stories is a critical point of our model. They were defined from the information contained in the user story, from the business logic of the application, from the source code, from the data flow, and the dependencies in the data model. For larger applications that have many user stories, it can be a complex task to determine these dependencies.

The problem of assigning user stories to microservices has an NP-hard complexity, when the number of user stories increases the runtime of the genetic algorithm increases considerably. The average time of execution in the tests carried out did not exceed 10 minutes, using a core-i7 computer, with 16 gigabytes of Ram (a population of 1000 individuals, convergence 10%, the maximum number of iterations 400 with 500 children and 500 mutations in each iteration).

The genetic algorithm is not deterministic, in each execution it can give different results, to reduce this problem, we executed the algorithm several times, we selected for each case the best solution, and we verified that it was repeated most of the times.

An algorithm was implemented that calculates the evaluation metrics; the same algorithm was used for calculating the metrics of all the decompositions used for comparison.

The “Lack of cohesion” metric is calculated from the interdependence of the microservices, it depends on the number of microservices. We concluded that MB proposes high cohesion solutions because their semantic similarity is high, so the microservices refer to the same topic, subject, or entity; semantic cohesion was proposed by other authors [27], [62].

VII. CONCLUSIONS AND FUTURE WORK

Microservices Backlog model (MB) allows architects, designers, or developers to reasoning about microservice granularity at design time, they can analyze metrics, diagrams, and dependencies of the microservices; they can notice critical points, estimated development time of the application; they can test different solutions or decomposition, analyze them, and select the better to be implemented.

The development team can evaluate different ways of distributing the user stories in microservices and take decisions based on metrics, graphs, and comparative analysis at design time. Therefore, using the MB model is possible to reason about the granularity of microservices at design time,

thus filling one of the research gaps proposed in the literature review.

The distribution of user stories in microservices affects coupling, cohesion, complexity, impacting the performance, modularity, and maintainability of the microservice-based application. Associating a user story to a single microservice (finer granularity), following the simple responsibility principle, implies more coupling to the application, equally having a monolithic application is not the best option in terms of maintainability, scalability, testing, and deployment of the application. The optimal solution is somewhere in between these two and depends on the functional requirements and characteristics of the application, the development team and the non-functional requirements to be addressed.

When comparing MB with the related works, none of the identified works used user stories as input data, none used data from agile practices or agile software development. MB model obtained low coupling, low or similar lack of cohesion, small calls, and low complexity than the state-of-the-art approaches; therefore, using our model, the software architect or development team can obtain microservices-based applications with low coupling, low complexity, and fewer calls between microservices.

Unlike other proposed works, one of the identified works used performance, functionality, maintainability, and modularity at the same time to evaluate the granularity of the microservices as MB. Only one paper used quality attributes as runtime characteristics (i.e. scalability, performance) and at the same time software as an artifact characteristic (i.e. modularity, maintainability). No papers addressed functionality, performance, modularity, and maintainability at the same time.

MB covers both aspects of runtime characteristics and software as artifact characteristics and MB uses coupling, cohesion, and complexity metrics to evaluate the candidate microservices of the application. Therefore, this research work fills proposed research gaps in the state of the art and represents a novel proposal to the development of microservice-based applications.

Moreover, according to limitations we propose the future work as follow: we will build a dataset of the reported microservices projects, identifying user stories and dependencies for a deeper validation, we will propose an automatic method of determining dependencies among user stories; we will implement the genetic algorithm using parallel programming to improve the runtime; we will review and propose another cohesion metric, include it in the model and evaluate the results; and we will generate source code or templates for the selected solution, as well as estimate computational resources, and deployment options for the microservices-based application.

REFERENCES

[1] K. Beck and M. Fowler, *Planning Extreme Programming*. Addison Wesley, 2001.

[2] M. Pikkarainen, J. Haikara, O. Salo, P. Abrahamsson, and J. Still, "The impact of agile practices on communication in software development," *Empir. Softw. Eng.*, vol. 13, no. 3, pp. 303–337, Jun. 2008, doi: 10.1007/s10664-008-9065-9.

[3] Versionone Enterprise, I. Digital.ai Software, and I. CollabNet, "14th annual state of agile report," 2019. [Online]. Available: www.stateofagile.com.

[4] T. Sedano, P. Ralph, and C. Peraire, "The Product Backlog," in *Proceedings - International Conference on Software Engineering*, May 2019, vol. 2019-May, pp. 200–211, doi: 10.1109/ICSE.2019.00036.

[5] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, 2016, doi: 10.1109/MS.2016.64.

[6] O. Zimmermann, "Microservices tenets: Agile approach to service development and deployment," *Comput. Sci. - Res. Dev.*, vol. 32, no. 3–4, pp. 301–310, 2017, doi: 10.1007/s00450-016-0337-0.

[7] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May 2018, doi: 10.1109/MS.2018.2141039.

[8] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice Transition and its Granularity Problem: A Systematic Mapping Study," *Softw. Pract. Exp.*, no. February, pp. 1–31, 2020, doi: 10.1002/spe.2869.

[9] N. Kulkarni and V. Dwivedi, "The role of service granularity in a successful soa realization - A case study," in *Proceedings - 2008 IEEE Congress on Services, SERVICES 2008*, 2008, vol. PART 1, pp. 423–430, doi: 10.1109/SERVICES-1.2008.86.

[10] A. Homay, M. de Sousa, A. Zoitl, and M. Wollschlaeger, "Service Granularity in Industrial Automation and Control Systems," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2020, pp. 132–139, doi: 10.1109/ETFA46521.2020.9212048.

[11] F. H. Vera-Rivera, H. Astudillo, and C. M. Gaona-Cuevas, "Defining and measuring microservice granularity – a literature overview," *PeerJ Comput. Sci.*, vol. In review.

[12] F. H. Vera-Rivera, E. G. Puerto-Cuadros, H. Astudillo, and C. M. Gaona-Cuevas, "Microservices Backlog - A Model of Granularity Specification and Microservice Identification," in *International conference on service computing SCC 2020. Lecture Notes in Computer Science*, Jun. 2020, vol. 12409 LNCS, pp. 85–102, doi: 10.1007/978-3-030-59592-0_6.

[13] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service Cutter: A Systematic Approach to Service Decomposition," in *IFIP International Federation for Information Processing 2016*, 2016, pp. 185–200, doi: 10.1007/978-3-319-44482-6_12.

[14] C. Richardson and microservices.io, "Microservice Architecture pattern." https://microservices.io/patterns/microservices.html (accessed Dec. 12, 2019).

[15] O. Zimmermann, M. Stocker, U. Zdun, D. Lübke, and C. Pautasso, "Microservice API Patterns," 2019. https://www.microservice-api-patterns.org/introduction (accessed Dec. 17, 2019).

[16] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kroger, "Microservice Decomposition via Static and Dynamic Analysis of the Monolith," *Proc. - 2020 IEEE Int. Conf. Softw. Archit. Companion, ICSA-C 2020*, pp. 9–16, 2020, doi: 10.1109/ICSA-C50368.2020.00011.

[17] O. Al-Debagy and P. Martinek, "Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach," in *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Jun. 2020, pp. 289–294, doi: 10.1109/SoSE50414.2020.9130466.

[18] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems based on Execution Traces," *IEEE Trans. Softw. Eng.*, vol. X, no. X, pp. 1–1, 2019, doi: 10.1109/TSE.2019.2910531.

[19] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning

approach for web application auto-decomposition into microservices," *J. Syst. Softw.*, vol. 151, pp. 243–257, 2019, doi: 10.1016/j.jss.2019.02.031.

[20] S. Li *et al.*, "A dataflow-driven approach to identifying microservices from monolithic applications," *J. Syst. Softw.*, vol. 157, 2019, doi: 10.1016/j.jss.2019.07.008.

[21] D. Taibi and K. Syst, "From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining," *Int. Conf. Cloud Comput. Serv. Sci. - CLOSER 2019*, no. March, 2019.

[22] N. Santos *et al.*, "A logical architecture design method for microservices architectures," in *ACM International Conference Proceeding Series*, Sep. 2019, vol. 2, pp. 145–151, doi: 10.1145/3344948.3344991.

[23] O. Al-Debagy and P. Martinek, "A new decomposition method for designing microservices," *Period. Polytech. Electr. Eng. Comput. Sci.*, vol. 63, no. 4, pp. 274–281, 2019, doi: 10.3311/PPee.13925.

[24] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Business object centric microservices patterns," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, vol. 11877 LNCS, pp. 476–495, doi: 10.1007/978-3-030-33246-4_30.

[25] L. Nunes, N. Santos, and A. Rito Silva, "From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts," in *13th European Conference, ECSA 2019. Lectures Notes in Computer Science 11681*, 2019, pp. 37–52, doi: 10.1007/978-3-030-29983-5_3.

[26] A. Homay, A. Zoitl, M. De Sousa, M. Wollschlaeger, and C. Chrysoulas, "Granularity cost analysis for function block as a service," *IEEE Int. Conf. Ind. Informatics*, vol. 2019-July, pp. 1199–1204, 2019, doi: 10.1109/INDIN41052.2019.8972205.

[27] M. Cojocaru, A. Uta, and A. M. Oprescu, "MicroValid: A validation framework for automatically decomposed microservices," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, Dec. 2019, vol. 2019-Decem, pp. 78–86, doi: 10.1109/CloudCom.2019.00023.

[28] A. Christoforou, L. Odysseos, and A. Andreou, "Migration of Software Components to Microservices: Matching and Synthesis," in *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2019, pp. 134–146, doi: 10.5220/0007732101340146.

[29] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search," in *17th International Conference Service-Oriented Computing. Lectures Notes in computer science 11895*, Oct. 2019, pp. 58–63, doi: 10.1007/978-3-030-33702-5_5.

[30] M. I. Josélyne, D. Tuheirwe-Mukasa, B. Kanagwa, and J. Balikuddembe, "Partitioning microservices - A Domain Engineering Approach," in *Proceedings of the 2018 International Conference on Software Engineering in Africa - SEiA '18*, 2018, pp. 43–49, doi: 10.1145/3195528.3195535.

[31] H. Vural, M. Koyuncu, and S. Misra, "A Case Study on Measuring the Size of Microservices," in *International Conference on Computational Science and Its Applications - ICCSA 2018*, 2018, pp. 454–463, doi: 10.1007/b98054.

[32] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, "Identifying Microservices Using Functional Decomposition," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, 2018, vol. 10998, pp. 50–65, doi: 10.1007/978-3-319-99933-3.

[33] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, "Function-Splitting Heuristics for Discovery of Microservices in Enterprise Systems," 2018, pp. 37–53.

[34] M. Tusjunt and W. Vatanawood, "Refactoring Orchestrated Web Services into Microservices Using Decomposition Pattern," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, Dec. 2018, pp. 609–613, doi: 10.1109/CompComm.2018.8781036.

[35] Z. Ren *et al.*, "Migrating web applications from monolithic structure to microservices architecture," in *ACM International Conference Proceeding Series*, Sep. 2018, pp. 1–10, doi: 10.1145/3275219.3275230.

[36] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 2017, pp. 243–246, doi: 10.1109/ICSAW.2017.11.

[37] J. P. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture," in *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, 2017, pp. 62–65, doi: 10.1109/ICSAW.2017.35.

[38] D. Shadija, M. Rezai, and R. Hill, "Microservices: Granularity vs. Performance," in *UCC '17 Companion Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, 2017, pp. 215–220, doi: 10.1145/3147234.3148093.

[39] S. Hassan, N. Ali, and R. Bahsoon, "Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity," in *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, Apr. 2017, pp. 1–10, doi: 10.1109/ICSA.2017.32.

[40] L. Baresi, M. Garriga, and A. De Renzis, *Microservices Identification through Interface Analysis*, vol. 10465, no. November. Cham: Springer International Publishing, 2017.

[41] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, Jun. 2017, pp. 524–531, doi: 10.1109/ICWS.2017.61.

[42] G. Kecskemeti, A. Kertesz, and A. C. Marosi, "Towards a methodology to form microservices from monolithic ones," in *Euro-Par 2016 Workshops - Lecture Notes in Computer Science*, 2017, vol. 10104 LNCS, pp. 284–295, doi: 10.1007/978-3-319-58943-5_23.

[43] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," *Proc. - 2016 IEEE Int. Conf. Serv. Comput. SCC 2016*, pp. 813–818, 2016, doi: 10.1109/SCC.2016.113.

[44] M. Ahmadvand and A. Ibrahim, "Requirements Reconciliation for Scalable and Secure Microservice (De)composition," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Sep. 2016, pp. 68–73, doi: 10.1109/REW.2016.026.

[45] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, 2004, Accessed: May 16, 2018. [Online]. Available: https://pdfs.semanticscholar.org/fa72/91f2073cb6fdbdd7c2213bf6d776d0ab411c.pdf.

[46] E. Evans, *Domain-Driven Design Reference - Definitions and Pattern Summaries*. 2015.

[47] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *European Conference on Service-Oriented and Cloud Computing - Lecture Notes in Computer Science.*, Sep. 2017, vol. 10465 LNCS, pp. 19–33, doi: 10.1007/978-3-319-67262-5_2.

[48] J. Bogner, S. Wagner, and A. Zimmermann, "Towards a practical maintainability quality model for service-and microservice-based systems," in *Proceedings of the 11th European Conference on Software Architecture Companion Proceedings - ECSA '17*, 2017, vol. 3, pp. 195–198, doi: 10.1145/3129790.3129816.

[49] I. Candela, G. Bavota, B. Russo, and R. Oliveto, "Using cohesion and coupling for software remodularization: Is it enough?," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, May 2016, doi: 10.1145/2928268.

[50] D. Rud, A. Schmietendorf, and R. R. Dumke, "Product Metrics for Service-Oriented Infrastructures," in *Conference: Applied Software Measurement. Proceedings of the International Workshop on Software Metrics and DASMA Software Metrik Kongress (IWSM/MetriKon 2006)*, 2006, Accessed: Jun. 18,

2019. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.122.6887&rep=rep1&type=pdf.

[51] Spacy.io, "Word Vectors and Semantic Similarity · spaCy Usage Documentation." https://spacy.io/usage/vectors-similarity#basics (accessed Nov. 20, 2020).

[52] M. Hirzalla, J. Cleland-Huang, and A. Arsanjani, "A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures," Springer, Berlin, Heidelberg, 2009, pp. 41–52.

[53] M. Cohn, *User Stories applied for agile software development*. Addison Wesley. Pearson Education Inc., 2004.

[54] M. Cohn, *Agile Estimating and Planning*. New York, NY, USA, 2005.

[55] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison Wesley, 2000.

[56] J. Holland, *Adaptation in natural and artificial systems*. Michigan: University of Michigan Press, 1975.

[57] F. Herrera, M. Lozano, and J. L. Verdegay, *Algoritmos Genéticos: Fundamentos, Extensiones y Aplicaciones*. ProQuest, 1995.

[58] mybatis.org, "Mybatis Jpetstore-6: A web application built on top of MyBatis 3, Spring 3 and Stripes." https://github.com/mybatis/jpetstore-6 (accessed Nov. 22, 2020).

[59] E. Evans, *Domain-Driven Design*. Addison Wesley, 2004.

[60] M. I. Rahman, S. Panichella, and D. Taibi, "A Curated Dataset of Microservices-Based Systems," in *Joint Proceedings of the Inforte Summer School on Software Maintenance and Evolution (CEUR Workshop Proceedings; Vol. 2520). CEUR-WS.*, 2019, vol. 2520, Accessed: Feb. 14, 2020. [Online]. Available: http://research.tuni.fi/clowee.

[61] G. Marquez and H. Astudillo, "Actual Use of Architectural Patterns in Microservices-Based Open Source Projects," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Jul. 2018, vol. 2018-December, pp. 31–40, doi: 10.1109/APSEC.2018.00017.

[62] M. Perepletchikov, C. Ryan, and K. Frampton, "Cohesion Metrics for Predicting Maintainability of Service-Oriented Software," in *Seventh International Conference on Quality Software (QSIC 2007)*, 2007, pp. 328–335, doi: 10.1109/QSIC.2007.4385516.